RenderingControl:2 Service Template Version 1.01

For UPnP™ Version 1.0
Status: Approved Standard
Date: May 31, 2006
Document Version: 1.00

Authors	Company
Alan Presser	Allegrosoft
Gary Langille	Echostar
Gerrie Shults	HP
John Ritchie (Co-Chair)	Intel
Mark Walker	Intel
Changhyun Kim	LG Electronics
Sungjoon Ahn	LG Electronics
Masatomo Hori	Matsushita Electric (Panasonic)
Matthew Ma	Matsushita Electric (Panasonic)
Jack Unverferth	Microsoft
Wim Bronnenberg	Philips
Geert Knapen (Co-Chair)	Philips
Russell Berkoff	Pioneer
Irene Shen	Pioneer
Norifumi Kikkawa	Sony

Authors	Company
Jonathan Tourzan	Sony
Yasuhiro Morioka	Toshiba

Contents

List of Tables

List of Figures

1 Overview and Scope

This service template is compliant with the UPnP Device Architecture version 1.0. It defines a service type referred to herein as RenderingControl.

1.1 Introduction

Most rendering devices contain a number of dynamically configurable attributes that affect how the current content is rendered. For example, video rendering devices, such as TVs, allow user control of display characteristics such as brightness and contrast, whereas audio rendering devices allow control of audio characteristics such as volume, balance, equalizer settings, etc. The RenderingControl service is intended to provide control points with the ability to query and/or adjust any rendering attribute that the device supports.

The RenderingControl service enables a control point to:

- Discover the set of attributes supported by the device.
- Retrieve the current setting of any supported attribute
- Change the setting of (that is: control) any modifiable attribute
- Restore the settings defined by a named Preset

The RenderingControl service DOES NOT:

- Control the flow of the associated content (for example, Play, Stop, Pause, Seek, etc.).
- Provide a mechanism to enumerate locally stored content.
- Provide a mechanism to select the content that is to be rendered.
- Provide a mechanism to send content to another device (via the home network or direct connection).

1.2 Multi-input Devices

Some high-end AV device are capable of receiving multiple pieces of content at the same time and combining that content together so that it can be rendered together using a single set of output hardware. For example, while displaying a TV program, high-end TVs can also display additional content (for example, VCR content) in a PIP (Picture-In-Picture) window. Similarly, a Karaoke machine can mix together the background music with a singer's voice so that both sounds are played together on the same set of speakers.

As with all devices, the RenderingControl service allows a control point to adjust the output characteristics of the post-mixed content before it is actually rendered. However, in many cases, control points may need to control the output characteristics of the individual input content before it is mixed together with the other input content. In order to support this, the RenderingControl service includes an *InstanceID* argument with each action that allows the control point to identify on which content the action is to be applied (for example, the post-mixed content or one of the pre-mixed input content items).

By convention, an *InstanceID* of 0 indicates that the invoked action MUST be applied to the post-mixed content. Similarly, each pre-mixed input content is assigned a unique *InstanceID* whose value is a non-zero, positive integer. Refer to Section 2.5, "Theory of Operation" for additional information.

1.3 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 [RFC 2119].

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*.
- Keywords that are defined by the UPnP AV 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, parentProperty::childProperty.

1.3.1 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 [DEVICE]. The XML Schema namespace is used to define property data types [XML SCHEMA-2].

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 as input properties, the values "*false*", "*true*" may also be encountered and MUST be accepted. Nevertheless, it is strongly RECOMMENDED that all properties be represented as "*0*" and "*1*".

1.3.2 Strings Embedded in Other Strings

Some string variables and arguments 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 embedded strings in two contexts – Comma Separated Value (CSV) lists (see Section 1.4.1, "Comma Separated Value (CSV) Lists") and property values in search criteria strings. Escaping conventions use the backslash character, "\" (character code U+005C), as follows:

a. Backslash ("\") is represented as "\\" in both contexts.
b. Comma (",") is
    1. represented as "\," in individual substring entries in CSV lists

2. not escaped in search strings

c. Double quote (“"”) is
   1. not escaped in CSV lists
   2. not escaped in search strings when it appears as the start or end delimiter of a property value
   3. represented as “\"” in search strings when it appears as a character that is part of the property value

1.3.3 Extended Backus-Naur Form

Extended Backus-Naur Form is used in this document for a formal syntax description of certain constructs. The usage here is according to the reference [EBNF].

1.3.3.1 Typographic conventions for EBNF

`Non-terminal` symbols are unquoted sequences of characters from the set of English upper and lower case letters, the digits “`0`” through “`9`”, and the hyphen (“`-`”). Character sequences between `'single quotes'` are terminal strings and MUST appear literally in valid strings. Character sequences between `(*comment delimiters*)` are English language definitions or supplementary explanations of their associated symbols. White space in the EBNF is used to separate elements of the EBNF, not to represent white space in valid strings. White space usage in valid strings is described explicitly in the EBNF. Finally, the EBNF uses the following operators:

Table 1-1: EBNF Operators

Operator	Semantics
`::=`	**definition** – the non-terminal symbol on the left is defined by one or more alternative sequences of terminals and/or non-terminals to its right.
`|`	**alternative separator** – separates sequences on the right that are independently allowed definitions for the non-terminal on the left.
`*`	**null repetition** – means the expression to its left MAY occur zero or more times.
`+`	**non-null repetition** – means the expression to its left MUST occur at least once and MAY occur more times.
`[ ]`	**optional –** the expression between the brackets is optional.
`( )`	**grouping –** groups the expressions between the parentheses.
`-`	**character range** – represents all characters between the left and right character operands inclusively.

1.4 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 property xsd:string data types. The following definition applies to both string data types.

1.4.1 Comma Separated Value (CSV) Lists

The UPnP AV services use state variables, action arguments and properties that represent lists – or one-dimensional arrays – of values. The UPnP Device Architecture, Version 1.0 [DEVICE], 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 (*x*, *y*, *z*), where *x*, *y* and *z* are the types of the individual values. 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 ({*x*, *y*, *z*}), where *x*, *y* and *z* are the types of the individual values in the subsequence and the subsequence MAY be repeated zero or more times.

- A list is represented as a **string** type (for state variables and action arguments) or xsd:string type (for properties).
- Commas separate values within a list.
- Integer values are represented in CSVs with the same syntax as the integer data type specified in [DEVICE] (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 [DEVICE]: **0**, **false**, **no**, **1**, **true**, **yes**.
- Boolean values are represented in property 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 1.3.2, "Strings Embedded in Other Strings".
- 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 1-2: CSV Examples

Type refinement of string	Value	Comments
CSV (**string**) or CSV (xsd:string)	"+artist,-date"	List of 2 property sort criteria.
CSV (**int**) or CSV (xsd:integer)	"1,-5,006,0,+7"	List of 5 integers.
CSV (**boolean**) or CSV (xsd:Boolean)	"0,1,1,0"	List of 4 booleans
CSV (**string**) or CSV (xsd:string)	"Smith\, Fred,Jones\, Davey"	List of 2 names, "Smith, Fred" and "Jones, Davey"
CSV (**i4**,**string**,**ui2**) or CSV (xsd:int, xsd:string, xsd:unsignedShort)	"-29837, string with leading blanks,0"	Note that the second value is " string with leading blanks"
CSV (**i4**) or CSV (xsd:int)	"3, 4"	Illegal CSV. White space is not allowed as part of an integer value.
CSV (**string**) or CSV (xsd:string)	",,"	List of 3 empty string values

Type refinement of string	Value	Comments
CSV (heterogeneous)	“Alice,Marketing,5,Sue,R&D,21,Dave,Finance,7”	List of unspecified number of people and associated attributes. Each person is described by 3 elements: a name **string**, a department **string** and years-of-service **ui2** or a name xsd:string, a department xsd:string and years-of-service xsd:unsignedShort.

1.5 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 AV object properties are represented in XML by element and attribute names, therefore, all property names belong to an XML namespace.

For the same reason that namespace prefixes are convenient in XML documents, it is convenient in specification text to refer to namespaces using a namespace prefix. Therefore, this specification declares a “standard” prefix for all XML namespaces used herein. In addition, this specification expands the scope where these prefixes have meaning, beyond a single XML document, to all of its text, XML examples, and certain string-valued properties. This expansion of scope *does not* supercede XML rules for usage in documents, it only augments and complements them in important contexts that are out-of-scope for the XML specifications.

All of the namespaces used in this specification are listed in the Tables “Namespace Definitions” and “Schema-related Information”. For each such namespace, Table 1-3, “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 1-4, “Schema-related Information”, to cross-reference additional namespace information. This second table includes each namespace’s valid XML document root elements (if any), its schema file name, versioning information (to

be discussed in more detail below), and links to the entries in the Reference section 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.

The Working Committee expects to continue refining these schemas after specification release to reduce the number of documents that are validated by the schemas while violating the specifications, but the schemas will still be informative, supporting documents. Some schemas might become normative in future versions of the specifications.

Table 1-3: Namespace Definitions

Standard Name-space Prefix	Namespace Name	Namespace Description	Normative Definition Document Reference
AV Working Committee defined namespaces			
av:	urn:schemas-upnp-org:av:av	Common data types for use in AV schemas	[AV-XSD]
avs:	urn:schemas-upnp-org:av:avs	Common structures for use in AV schemas	[AVS-XSD]
avdt:	urn:schemas-upnp-org:av:avdt	Datastructure Template	[AVDT]
avt-event:	urn:schemas-upnp-org:metadata-1-0/AVT/	Evented *LastChange* state variable for AVTransport	[AVT]
didl-lite:	urn:schemas-upnp-org:metadata-1-0/DIDL-Lite/	Structure and metadata for ContentDirectory	[CDS]
rcs-event:	urn:schemas-upnp-org:metadata-1-0/RCS/	Evented *LastChange* state variable for RenderingControl	[RCS]
srs:	urn:schemas-upnp-org:av:srs	Metadata and structure for ScheduledRecording	[SRS]
srs-event:	urn:schemas-upnp-org:av:srs-event	Evented *LastChange* state variable for ScheduledRecording	[SRS]
upnp:	urn:schemas-upnp-org:metadata-1-0/upnp/	Metadata for ContentDirectory	[CDS]
Externally defined namespaces			
dc:	http://purl.org/dc/elements/1.1/	Dublin Core	[DC-TERMS]
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 1-4: Schema-related Information

Standard Name-space Prefix	Relative URI and File Name • Form 1 • Form 2	Valid Root Element(s)	Schema Reference
AV Working Committee Defined Namespaces			
av:	• av-v*n-yyyymmdd*.xsd • av-v*n*.xsd	*n/a*	[AV-XSD]

Standard Name-space Prefix	Relative URI and File Name • Form 1 • Form 2	Valid Root Element(s)	Schema Reference
avs:	• avs-v*n-yyyymmdd*.xsd • avs-v*n*.xsd	`<Features>` `<stateVariableValuePairs>`	[AVS-XSD]
avdt:	• avdt-v*n-yyyymmdd*.xsd • avdt-v*n*.xsd	`<AVDT>`	[AVDT]
avt-event:	• avt-event-v*n-yyyymmdd*.xsd • avt-event-v*n*.xsd	`<Event>`	[AVT-EVENT-XSD]
didl-lite:	• didl-lite-v*n-yyyymmdd*.xsd • didl-lite-v*n*.xsd	`<DIDL-Lite>`	[DIDL-LITE-XSD]
rcs-event:	• rcs-event-v*n-yyyymmdd*.xsd • rcs-event-v*n*.xsd	`<Event>`	[RCS-EVENT-XSD]
srs:	• srs-v*n-yyyymmdd*.xsd • srs-v*n*.xsd	`<srs>`	[SRS-XSD]
srs-event:	• srs-event-v*n-yyyymmdd*.xsd • srs-event-v*n*.xsd	`<StateEvent>`	[SRS-EVENT-XSD]
upnp:	• upnp-v*n-yyyymmdd*.xsd • upnp-v*n*.xsd	*n/a*	[UPNP-XSD]
Externally Defined Namespaces			
dc:	*Absolute URL*: http://dublincore.org/schemas/xmls/simpledc20021212.xsd		[DC-XSD]
xsd:	*n/a*	`<schema>`	[XMLSCHEMA-XSD]
xsi:	*n/a*		*n/a*
xml:	*n/a*		[XML-XSD]

1.5.1 Namespace Prefix Requirements

There are many occurrences in this specification of string data types that contain XML names (property names). These XML names in strings will not be processed under namespace-aware conditions. Therefore, all occurrences in instance documents of XML names in strings MUST use the standard namespace prefixes as declared in Table 1-3. In order to properly process the XML documents described herein, control points and devices MUST use namespace-aware XML processors [XML-NMSP] for both reading and writing. As allowed by [XML-NMSP], the namespace prefixes used in an instance document are at the sole discretion of the document creator. Therefore, the declared prefix for a namespace in a document MAY be different from the standard prefix. All devices MUST be able to correctly process any valid XML instance document, even when it uses a non-standard prefix for ordinary XML names. It is strongly RECOMMENDED that all devices use these standard prefixes for all instance documents to avoid confusion on the part of both human and machine readers. These standard prefixes are used in all descriptive text and all XML examples in this and related UPnP specifications. Also, each individual specification may assume a default namespace for its descriptive text. In that case, names from that namespace may appear with no prefix.

The assumed default namespace, if any, for each UPnP AV specification is given in Table 1-5, “Default Namespaces for the AV Specifications”.

Note: all UPnP AV schemas declare attributes to be “unqualified”, so namespace prefixes are never used with AV Working Committee defined attribute names.

Table 1-5: Default Namespaces for the AV Specifications

AV Specification Name	Default Namespace Prefix
AVTransport:2	avt-event:
ConnectionManager:2	*n/a*
ContentDirectory:2	didl-lite:
MediaRenderer:2	*n/a*
MediaServer:2	*n/a*
RenderingControl:2	rcs-event:
ScheduledRecording:1	srs:

1.5.2 Namespace Names, Namespace Versioning and Schema Versioning

Each namespace that is defined by the AV Working Committee is named by a URN.

In order to enable both forward and backward compatibility, the UPnP TC has established the general policy that namespace names will not change with new versions of specifications, even when the specification changes the definition of a namespace. But, namespaces still have version numbers that reflect definitional changes. Each time the definition of a namespace is changed, the namespace's version number is incremented by one. Therefore, namespace version information must be provided with each XML instance document so that the document's receiver can properly understand its meaning. This is achieved by the following rules:

- Every release of a schema is identified by a version number and date of the form "*n-yyyymmdd*", where *n* corresponds to the namespace definition version number and *yyyymmdd* is the year, month and day in the Gregorian calendar that the schema is released.

  For example, the new version numbers of the pre-existing "DIDL-Lite" and "upnp" schemas are "2". Versions for new schemas, such as "srs" are "1".

  For each schema, the version-date will appear in two places:

  1. In the schema file name, according to the naming structure shown in Table 1-4, "Schema-related Information".
  2. As the value of the `version` attribute of each schema's `schema` root element.

Namespaces are referenced in both schema and XML instance documents by namespace name. The namespace name appears as the value of an `xmlns` attribute. The `xmlns` attribute also declares a namespace prefix that will be used to qualify names from each namespace. Schemas are referenced in both schema and XML instance documents by URI in the `schemaLocation` attribute. See section 1.5.3, "Namespace Usage Examples" . Two different forms of URI are available, each with a different meaning. All UPnP AV-defined schema URIs share a common base path of "http://www.upnp.org/schemas/av/". Each schema URI has two unique relative forms (see Table 1-4, "Schema-related Information"), according to which version of a namespace and its representative schema is of interest. The allowed relative URI forms are:

1. *schema-root-name* "-v" *version-date*
   where *version-date* is a full version-date of the form *n-yyyymmdd*. This form references the schema whose "root" name (typically the standardized prefix name used for the namespace that the schema represents) and version-date match *schema-root-name* and *version-date*, respectively.
2. *schema-root-name* "-v" *version*
   where *version* is an integer representing the namespace's version number. This form references the most recent version of the schema whose root name and namespace version number match *schema-root-name* and the *version*, respectively.

Usage rules for schema location URIs are as follows:

- All instance documents, whether generated by a service or a control point, MUST use Form 1.
- All UPnP AV published schemas that reference other UPnP AV schemas will also use Form 1.
- Validation of XML instance documents in UPnP AV systems potentially serves two purposes. The first is based on standard XML and XML Schema semantics: the document's creator asserts that the document is syntactically correct with respect to the referenced schema. The receiving processor can confirm this with a validating parser that uses the referenced schema(s). The second is based on UPnP AV namespace semantics. The receiving processor knows that the XML instance document is supposed to conform to one or more specific UPnP AV specifications. Since the second context is actually the more important context for instance document processing, the receiving processor MAY validate the instance document against any version of a schema that satisfies its needs in assessing the acceptability of the received instance document.

1.5.3 Namespace Usage Examples

The `schemaLocation` attribute for XML instance documents comes from the XML Schema instance 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: 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 *DIDL-Lite XML Document*. This document assumes version-date 2-20060531 of the "didl-lite:" namespace/schema combination and (a possible later) version 2-20061231 of "upnp:". The lines with the gray background show how to express this versioning information in the instance document.

```
<?xml version="1.0" encoding="UTF-8"?>
<DIDL-Lite
 xmlns:dc="http://purl.org/dc/elements/1.1/"
 xmlns="urn:schemas-upnp-org:metadata-1-0/DIDL-Lite/"
 xmlns:upnp="urn:schemas-upnp-org:metadata-1-0/upnp/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="
   urn:schemas-upnp-org:metadata-1-0/DIDL-Lite/
    http://www.upnp.org/schemas/av/didl-lite-v2-20060531.xsd
   urn:schemas-upnp-org:metadata-1-0/upnp/
    http://www.upnp.org/schemas/av/upnp-v2-20061231.xsd">
   <item id="18" parentID="13" restricted="0">
      ...
   </item>
</DIDL-Lite>
```

Example 2:

Sample *srs XML Document*. This document assumes version 1-20060531 of the "srs:" namespace/schema combination. Again, the lines with the gray background show how to express this versioning information in the instance document.

```
<?xml version="1.0" encoding="UTF-8"?>
<srs
 xmlns="urn:schemas-upnp-org:av:srs"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="
   urn:schemas-upnp-org:av:srs
    http://www.upnp.org/schemas/av/srs-v1-20060531.xsd">
   ...
```

```
</srs>
```

1.6 Vendor-defined Extensions

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

1.7 References

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

[AVARCH] – *AVArchitecture:1*, UPnP Forum, June 25, 2002.
Available at: http://www.upnp.org/specs/av/UPnP-av-AVArchitecture-v1-20020625.pdf.

[AVDT] – *AV DataStructure Template:1*, UPnP Forum, May 31, 2006.
Available at: http://www.upnp.org/specs/av/UPnP-av-AVDataStructure-v1-20060531.pdf.
Latest version available at: http://www.upnp.org/specs/av/UPnP-av-AVDataStructure-v1.pdf.

[AVDT-XSD] – *XML Schema for UPnP AV Datastructure Template:1*, UPnP Forum, May 31, 2006.
Available at: http://www.upnp.org/schemas/av/avdt-v1-20060531.xsd.
Latest version available at: http://www.upnp.org/schemas/av/avdt-v1.xsd.

[AV-XSD] – *XML Schema for UPnP AV Common XML Data Types*, UPnP Forum, May 31, 2006.
Available at: http://www.upnp.org/schemas/av/av-v1-20060531.xsd.
Latest version available at: http://www.upnp.org/schemas/av/av-v1.xsd.

[AVS-XSD] – *XML Schema for UPnP AV Common XML Structures*, UPnP Forum, May 31, 2006.
Available at: http://www.upnp.org/schemas/av/avs-v1-20060531.xsd.
Latest version available at: http://www.upnp.org/schemas/av/avs-v1.xsd.

[AVT] – *AVTransport:2*, UPnP Forum, May 31, 2006.
Available at: http://www.upnp.org/specs/av/UPnP-av-AVTransport-v2-Service-20060531.pdf.
Latest version available at: http://www.upnp.org/specs/av/UPnP-av-AVTransport-v2-Service.pdf.

[AVT-EVENT-XSD] – *XML Schema for AVTransport:2 LastChange Eventing*, UPnP Forum, May 31, 2006.
Available at: http://www.upnp.org/schemas/av/avt-event-v2-20060531.xsd.
Latest version available at: http://www.upnp.org/schemas/av/avt-event-v2.xsd.

[CDS] – *ContentDirectory:2*, UPnP Forum, May 31, 2006.
Available at: http://www.upnp.org/specs/av/UPnP-av-ContentDirectory-v2-Service-20060531.pdf.
Latest version available at: http://www.upnp.org/specs/av/UPnP-av-ContentDirectory-v2-Service.pdf.

[CM] – *ConnectionManager:2*, UPnP Forum, May 31, 2006.
Available at: http://www.upnp.org/specs/av/UPnP-av-ConnectionManager-v2-Service-20060531.pdf.
Latest version available at: http://www.upnp.org/specs/av/UPnP-av-ConnectionManager-v2-Service.pdf.

[DC-XSD] – *XML Schema for UPnP AV Dublin Core*.
Available at: http://www.dublincore.org/schemas/xmls/simpledc20020312.xsd.

[DC-TERMS] – *DCMI term declarations represented in XML schema language*.
Available at: http://www.dublincore.org/schemas/xmls.

[DEVICE] – *UPnP Device Architecture, version 1.0*, UPnP Forum, June 13, 2000.
Available at: http://www.upnp.org/specs/architecture/UPnP-DeviceArchitecture-v1.0-20000613.htm.
Latest version available at: http://www.upnp.org/specs/architecture/UPnP-DeviceArchitecture-v1.0.htm.

[DIDL] – ISO/IEC CD 21000-2:2001**,** *Information Technology - Multimedia Framework - Part 2: Digital Item Declaration,* July 2001.

[DIDL-LITE-XSD] – *XML Schema for ContentDirectory:2 Structure and Metadata (DIDL-Lite)*, UPnP Forum, May 31, 2006.
Available at: http://www.upnp.org/schemas/av/didl-lite-v2-20060531.xsd.
Latest version available at: http://www.upnp.org/schemas/av/didl-lite-v2.xsd.

[EBNF] – ISO/IEC 14977, *Information technology - Syntactic metalanguage - Extended BNF*, December 1996.

[HTTP/1.1] – *HyperText Transport Protocol – HTTP/1.1*, R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee, June 1999.
Available at: http://www.ietf.org/rfc/rfc2616.txt.

IEC 61883] – *IEC 61883 Consumer Audio/Video Equipment – Digital Interface - Part 1 to 5*.
Available at: http://www.iec.ch.

[IEC-PAS 61883] – *IEC-PAS 61883 Consumer Audio/Video Equipment – Digital Interface - Part 6*.
Available at: http://www.iec.ch.

[ISO 8601] – *Data elements and interchange formats – Information interchange -- Representation of dates and times*, International Standards Organization, December 21, 2000.
Available at: ISO 8601:2000.

[MIME] – *IETF RFC 1341, MIME (Multipurpose Internet Mail Extensions)*, N. Borenstein, N. Freed, June 1992.
Available at: http://www.ietf.org/rfc/rfc1341.txt.

[MR] – *MediaRenderer:2*, UPnP Forum, May 31, 2006.
Available at: http://www.upnp.org/specs/av/UPnP-av-MediaRenderer-v2-Device-20060531.pdf.
Latest version available at: http://www.upnp.org/specs/av/UPnP-AV-MediaRenderer-v2-Device.pdf.

[MS] – *MediaServer:2*, UPnP Forum, May 31, 2006.
Available at: http://www.upnp.org/specs/av/UPnP-av-MediaServer-v2-Device-20060531.pdf.
Latest version available at: http://www.upnp.org/specs/av/UPnP-AV-MediaServer-v2-Device.pdf.

[RCS] – *RenderingControl:2*, UPnP Forum, May 31, 2006.
Available at: http://www.upnp.org/specs/av/UPnP-av-RenderingControl-v2-Service-20060531.pdf.
Latest version available at: http://www.upnp.org/specs/av/UPnP-av-RenderingControl-v2-Service.pdf.

[RCS-EVENT-XSD] –*XML Schema for RenderingControl:2 LastChange Eventing*, UPnP Forum, May 31, 2006.
Available at: http://www.upnp.org/schemas/av/rcs-event-v1-20060531.xsd.
Latest version available at: http://www.upnp.org/schemas/av/rcs-event-v1.xsd.

[RFC 1738] – *IETF RFC 1738, Uniform Resource Locators (URL)*, Tim Berners-Lee, et. Al., December 1994.
Available at: http://www.ietf.org/rfc/rfc1738.txt.

[RFC 2119] – *IETF RFC 2119, Key words for use in RFCs to Indicate Requirement Levels*, S. Bradner, 1997.
Available at: http://www.faqs.org/rfcs/rfc2119.html.

[RFC 2396] – *IETF RFC 2396, Uniform Resource Identifiers (URI): Generic Syntax*, Tim Berners-Lee, et al, 1998.
Available at: http://www.ietf.org/rfc/rfc2396.txt.

[RFC 3339] – *IETF RFC 3339, Date and Time on the Internet: Timestamps*, G. Klyne, Clearswift Corporation, C. Newman, Sun Microsystems, July 2002.
Available at: http://www.ietf.org/rfc/rfc3339.txt.

[RTP] – *IETF RFC 1889, Realtime Transport Protocol (RTP)*, H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson, January 1996.
Available at: http://www.ietf.org/rfc/rfc1889.txt.

[RTSP] – *IETF RFC 2326, Real Time Streaming Protocol (RTSP)*, H. Schulzrinne, A. Rao, R. Lanphier, April 1998.
Available at: http://www.ietf.org/rfc/rfc2326.txt.

[SRS] – *ScheduledRecording:1*, UPnP Forum, May 31, 2006.
Available at: http://www.upnp.org/specs/av/UPnP-av-ScheduledRecording-v1-Service-20060531.pdf.
Latest version available at: http://www.upnp.org/specs/av/UPnP-av-ScheduledRecording-v1-Service-20060531.pdf.

[SRS-XSD] – *XML Schema for ScheduledRecording:1 Metadata and Structure*, UPnP Forum, May 31, 2006.
Available at: http://www.upnp.org/schemas/av/srs-v1-20060531.xsd.
Latest version available at: http://www.upnp.org/schemas/av/srs-v1.xsd.

[SRS-EVENT-XSD] – *XML Schema for ScheduledRecording:1 LastChange Eventing*, UPnP Forum, May 31, 2006.
Available at: http://www.upnp.org/schemas/av/srs-event-v1-20060531.xsd.
Latest version available at: http://www.upnp.org/schemas/av/srs-event-v1.xsd.

[UAX 15] – *Unicode Standard Annex #15, Unicode Normalization Forms, version 4.1.0, revision 25*, M. Davis, M. Dürst, March 25, 2005.
Available at: http://www.unicode.org/reports/tr15/tr15-25.html.

[UNICODE COLLATION] – *Unicode Technical Standard #10, Unicode Collation Algorithm version 4.1.0*, M. Davis, K. Whistler, May 5, 2005.
Available at: http://www.unicode.org/reports/tr10/tr10-14.html.

[UPNP-XSD] – *XML Schema for ContentDirectory:2 Metadata*, UPnP Forum, May 31, 2006.
Available at: http://www.upnp.org/schemas/av/upnp-v2-20060531.xsd.
Latest version available at: http://www.upnp.org/schemas/av/upnp-v2.xsd.

[UTS 10] – *Unicode Technical Standard #10, Unicode Collation Algorithm, version 4.1.0, revision 14*, M. Davis, K. Whistler, May 5, 2005.
Available at: http://www.unicode.org/reports/tr10/tr10-14.html.

[UTS 35] – *Unicode Technical Standard #35, Locale Data Markup Language, version 1.3R1, revision 5*,.M. Davis, June 2, 2005.
Available at: http://www.unicode.org/reports/tr35/tr35-5.html.

[XML] – *Extensible Markup Language (XML) 1.0 (Third Edition)*, François Yergeau, Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, eds., W3C Recommendation, February 4, 2004.
Available at: http://www.w3.org/TR/2004/REC-xml-20040204.

[XML-NS] – *The "xml:" Namespace*, November 3, 2004.
Available at: http://www.w3.org/XML/1998/namespace.

[XML-XSD] – *XML Schema for the "xml:" Namespace*.
Available at: http://www.w3.org/2001/xml.xsd.

[XML-NMSP] – *Namespaces in XML*, Tim Bray, Dave Hollander, Andrew Layman, eds., W3C Recommendation, January 14, 1999.
Available at: http://www.w3.org/TR/1999/REC-xml-names-19990114.

[XML SCHEMA-1] – *XML Schema Part 1: Structures, Second Edition*, Henry S. Thompson, David Beech, Murray Maloney, Noah Mendelsohn, W3C Recommendation, 28 October 2004.
Available at: http://www.w3.org/TR/2004/REC-xmlschema-1-20041028.

[XML SCHEMA-2] – *XML Schema Part 2: Data Types, Second Edition*, Paul V. Biron, Ashok Malhotra, W3C Recommendation, 28 October 2004.
Available at: http://www.w3.org/TR/2004/REC-xmlschema-2-20041028.

[XMLSCHEMA-XSD] – *XML Schema for XML Schema*.
Available at: http://www.w3.org/2001/XMLSchema.xsd.

2 Service Modeling Definitions

2.1 Service Type

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

urn:schemas-upnp-org:service:*RenderingControl:2*

The shorthand RenderingControl is used herein to refer to this service type.

2.2 State Variables

Table 2-1: State Variables

Variable Name	R/O[1]	Data Type	Allowed Value	Default Value	Eng. Units
LastChange	*R*	**string**			
PresetNameList	*R*	**string**	CSV[2] (**string**)		
Brightness	*O*	**ui2**	See Table 2-2		
Contrast	*O*	**ui2**	See Table 2-3		
Sharpness	*O*	**ui2**	See Table 2-4		
RedVideoGain	*O*	**ui2**	See Table 2-5		
GreenVideoGain	*O*	**ui2**	See Table 2-6		
BlueVideoGain	*O*	**ui2**	See Table 2-7		
RedVideoBlackLevel	*O*	**ui2**	See Table 2-8		
GreenVideoBlackLevel	*O*	**ui2**	See Table 2-9		
BlueVideoBlackLevel	*O*	**ui2**	See Table 2-10		
ColorTemperature	*O*	**ui2**	See Table 2-11		
HorizontalKeystone	*O*	**i2**	See Table 2-12		
VerticalKeystone	*O*	**i2**	See Table 2-13		
Mute	*O*	**boolean**			
Volume[3]	*O*	**ui2**	See Table 2-14		
VolumeDB	*O*	**i2**	See Table 2-15		1/256 dB
Loudness	*O*	**boolean**			
A_ARG_TYPE_Channel	*R*	**string**	See Table 2-16		
A_ARG_TYPE_InstanceID	*R*	**ui4**			
A_ARG_TYPE_PresetName	*R*	**string**	See Table 2-17		
A_ARG_TYPE_DeviceUDN	*O*	**string**			
A_ARG_TYPE_ServiceType	*O*	**string**			
A_ARG_TYPE_ServiceID	*O*	**string**			
A_ARG_TYPE_ StateVariableValuePairs	*O*	**string**			

Variable Name	R/O[1]	Data Type	Allowed Value	Default Value	Eng. Units
A_ARG_TYPE_StateVariableList	*O*	**string**	CSV (**string**)		
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. CSV is defined more formally in the ContentDirectory service template.

[3] The *Volume* and *VolumeDB* state variables are defined as a pair. Therefore, each implementation of this service MUST either support both of them or support none of them. At all times, these two state variables MUST remain synchronized with each other (that is: both of them MUST always represent the same volume setting).

Table 2-2: allowedValueRange for *Brightness*

	Value	R/O
minimum	*0*	*R*
maximum	*Vendor defined*	*R*
step	*1*	*R*

Table 2-3: allowedValueRange for *Contrast*

	Value	R/O
minimum	*0*	*R*
maximum	*Vendor defined*	*R*
step	*1*	*R*

Table 2-4: allowedValueRange for *Sharpness*

	Value	R/O
minimum	*0*	*R*
maximum	*Vendor defined*	*R*
step	*1*	*R*

Table 2-5: allowedValueRange for *RedVideoGain*

	Value	R/O
minimum	*0*	*R*
maximum	*Vendor defined*	*R*
step	*1*	*R*

Table 2-6: allowedValueRange for *GreenVideoGain*

	Value	R/O
minimum	*0*	*R*
maximum	*Vendor defined*	*R*
step	*1*	*R*

Table 2-7: allowedValueRange for *BlueVideoGain*

	Value	R/O
minimum	*0*	*R*
maximum	*Vendor defined*	*R*
step	*1*	*R*

Table 2-8: allowedValueRange for *RedVideoBlackLevel*

	Value	R/O
minimum	*0*	*R*
maximum	*Vendor defined*	*R*
step	*1*	*R*

Table 2-9: allowedValueRange for *GreenVideoBlackLevel*

	Value	R/O
minimum	*0*	*R*
maximum	*Vendor defined*	*R*
step	*1*	*R*

Table 2-10: allowedValueRange for *BlueVideoBlackLevel*

	Value	R/O
minimum	*0*	*R*
maximum	*Vendor defined*	*R*
step	*1*	*R*

Table 2-11: allowedValueRange for *ColorTemperature*

	Value	R/O
minimum	*0*	*R*
maximum	*Vendor defined*	*R*
step	*1*	*R*

Table 2-12: allowedValueRange for *HorizontalKeystone*

	Value	R/O
minimum	*Vendor defined* (MUST be <= 0)	*R*

	Value	R/O
maximum	*Vendor defined*	*R*
Step	*1*	*R*

Table 2-13: allowedValueRange for *VerticalKeystone*

	Value	R/O
minimum	*Vendor defined* (MUST be <= 0)	*R*
maximum	*Vendor defined*	*R*
step	*1*	*R*

Table 2-14: allowedValueRange for *Volume*

	Value	R/O
minimum	0	*R*
maximum	*Vendor defined*	*R*
Step	*1*	*R*

Table 2-15: allowedValueRange for *VolumeDB*

	Value	R/O
minimum	*Vendor defined*	*R*
maximum	*Vendor defined*	*R*
step	*Vendor defined*	*O*

Table 2-16: allowedValueList for *A_ARG_TYPE_Channel*

Value	R/O
"*Master*"	*R*
"*LF*"	*O*
"*RF*"	*O*
"*CF*"	*O*
"*LFE*"	*O*
"*LS*"	*O*
"*RS*"	*O*
"*LFC*"	*O*
"*RFC*"	*O*
"*SD*"	*O*
"*SL*"	*O*
"*SR*"	*O*
"*T*"	*O*
"*B*"	*O*

Value	R/O
Vendor-defined	

Table 2-17: allowedValueList for *A_ARG_TYPE_PresetName*

Value	R/O
"*FactoryDefaults*"	*R*
"*InstallationDefaults*"	*O*
Vendor defined	

2.2.1 *LastChange*

This state variable is used exclusively for eventing purposes to allow clients to receive meaningful event notifications whenever the state of the device changes. The *LastChange* state variable identifies all of the state variables that have changed since the last time the *LastChange* state variable was evented. Refer to Section 2.3.1, "Event Model" for additional information.

The format of this state variable conforms to the XML schema described in [RCS-EVENT-XSD]. The following *rcs-event XML document* illustrates a typical example of the schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<Event
 xmlns="urn:schemas-upnp-org:metadata-1-0/RCS/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="
   urn:schemas-upnp-org:metadata-1-0/RCS/
    http://www.upnp.org/schemas/av/rcs-event-v1-20060531.xsd">
   <InstanceID val="0">
      <Brightness val="36"/>
      <Contrast val="54"/>
      ...
   </InstanceID>
   <InstanceID val="1">
      <Mute channel="Master" val="0"/>
      <Volume channel="CF" val="24"/>
      ...
   </InstanceID>
   ...
</Event>
```

As illustrated above, the *LastChange* state variable contains a single root element whose body contains one or more *InstanceID* elements, each corresponding to a (virtual) instance of the RenderingControl service, whose state has changed since the last time the *LastChange* state variable was evented. Each *InstanceID* element contains one or more state variable elements that identify all of the state variables within that instance that changed. Each state variable element contains the new (current) value of that state variable.

In the example above, the *Brightness* and *Contrast* setting of instance 0 has changed to 36 and 54, respectively. Additionally, the *Mute* setting on the *Master* channel of instance 1 has been set to 0 (false) (that is: muting has been turned off) and the *Volume* of the Center Front channel of instance 1 has been set to 24.

Note: Only the audio-related state variables include a *channel* attribute, which identifies the audio channel that has experienced a change.

When a given state variable (within the same instance) changes multiple times before the moderation period of the *LastChange* state variable expires, only one state variable element for the affected state

variable will appear within the *InstanceID* element. The previous state variable element for the affected state variable MUST be removed and replaced with a new state variable element that reflects the most recent value of that state variable.

State variable elements MAY appear in any order within a given *InstanceID* element. This implies that no meaning may be deduced from the order in which the state variables for a given instance are listed. Similarly, the order of *InstanceID* elements has no particular meaning and they MAY appear in any order.

For example, when the *Brightness* of instance 0 changes from 26 to 54 then to 48, and the *Brightness* of instance 1 changes from 54 to 35, then to 11, *LastChange* is set to the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<Event
 xmlns="urn:schemas-upnp-org:metadata-1-0/RCS/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="
   urn:schemas-upnp-org:metadata-1-0/RCS/
    http://www.upnp.org/schemas/av/rcs-event-v1-20060531.xsd">
   <InstanceID val="0">
      <Brightness val="48"/>
   </InstanceID>
   <InstanceID val="1">
      <Brightness val="11"/>
   </InstanceID>
</Event>
```

The *LastChange* state variable is the only state variable that is evented using the standard UPnP event mechanism. All other state variables are indirectly evented via the *LastChange* state variable event. Refer to Section 2.3.1, "Event Model" for additional details.

Note that the LastChange state variable is XML and therefore needs to be escaped using the normal XML rules (see [XML] – Section 2.4 Character Data) before being transmitted as an event.

2.2.2 *PresetNameList*

This state variable contains a comma-separated list of valid preset names currently supported by this device. Its value changes if/when the device changes the set of presets that it supports. This may occur in conjunction with a vendor-defined action or some other non-UPnP event. This state variable will include any of the predefined presets that are supported by the device.

2.2.3 *Brightness*

This unsigned integer state variable represents the current brightness setting of the associated display device. Its value ranges from a minimum of 0 to some device specific maximum. A numerical change of 1 corresponds to the smallest incremental change that is supported by the device.

Higher values increase the brightness of the display's output.

2.2.4 *Contrast*

This unsigned integer state variable represents the current contrast setting of the associated display device. Its value ranges from a minimum of 0 to some device specific maximum. A numerical change of 1 corresponds to the smallest incremental change that is supported by the device.

Higher values increase the contrast of the display's output.

2.2.5 *Sharpness*

This unsigned integer state variable represents the current sharpness setting of the associated display device. Its value ranges from a minimum of 0 to some device specific maximum. A numerical change of 1 corresponds to the smallest incremental change that is supported by the device.

Higher values accentuate fine detail.

2.2.6 *RedVideoGain*

This unsigned integer state variable represents the current setting of the red gain control for the associated display device. Its value ranges from a minimum of 0 to some device specific maximum. A numerical change of 1 corresponds to the smallest incremental change that is supported by the device.

Higher values increase the intensity of red in the display's output. See Section 2.2.27.1, "*xxxVideoGain* and *xxxVideoBlackLevel*" for additional information.

2.2.7 *GreenVideoGain*

This unsigned integer state variable represents the current setting of the green gain control for the associated display device. Its value ranges from a minimum of 0 to some device specific maximum. A numerical change of 1 corresponds to the smallest incremental change that is supported by the device.

Higher values increase the intensity of green in the display's output. See Section 2.2.27.1, "*xxxVideoGain* and *xxxVideoBlackLevel*" for additional information.

2.2.8 *BlueVideoGain*

This unsigned integer state variable represents the current setting of the blue gain control for the associated display device. Its value ranges from a minimum of 0 to some device specific maximum. A numerical change of 1 corresponds to the smallest incremental change that is supported by the device.

Higher values increase the intensity of blue in the display's output. See Section 2.2.27.1, "*xxxVideoGain* and *xxxVideoBlackLevel*" for additional information.

2.2.9 *RedVideoBlackLevel*

This unsigned integer state variable represents the current setting for the minimum output intensity of red for the associated display device. Its value ranges from a minimum of 0 to some device specific maximum. A numerical change of 1 corresponds to the smallest incremental change that is supported by the device.

Higher values increase the minimum output intensity of red in the display's output. See Section 2.2.27.1, "*xxxVideoGain* and *xxxVideoBlackLevel*" for additional information.

2.2.10 *GreenVideoBlackLevel*

This unsigned integer state variable represents the current setting for the minimum output intensity of green for the associated display device. Its value ranges from a minimum of 0 to some device specific maximum. A numerical change of 1 corresponds to the smallest incremental change that is supported by the device.

Higher values increase the minimum output intensity of green in the display's output. See Section 2.2.27.1, "*xxxVideoGain* and *xxxVideoBlackLevel*" for additional information.

2.2.11 *BlueVideoBlackLevel*

This unsigned integer state variable represents the current setting for the minimum output intensity of blue for the associated display device. Its value ranges from a minimum of 0 to some device specific maximum. A numerical change of 1 corresponds to the smallest incremental change that is supported by the device.

Higher values increase the minimum output intensity of blue in the display's output. See Section 2.2.27.1, "*xxxVideoGain* and *xxxVideoBlackLevel*" for additional information.

2.2.12 *ColorTemperature*

This unsigned integer state variable represents the current setting for the color quality of white for the associated display device. Its value ranges from a minimum of 0 to some device specific maximum. A numerical change of 1 corresponds to the smallest incremental change that is supported by the device

Lower values produce warmer shades of white (that is: biased towards yellow/orange) and higher values produce cooler shades of white (that is: biased towards blue).

2.2.13 *HorizontalKeystone*

This signed integer state variable represents the current level of compensation for horizontal distortion (described below) of the associated display device. Its value ranges from device-specific negative number to a device specific positive number. Zero does not need to be in the middle of this range, although it will be for most devices. A numerical change of 1 corresponds to the smallest incremental change that is supported by the device.

Horizontal distortion can occur when the display device is horizontally misaligned from the center of the viewing screen. For example, when a video/still-image projection device is shifted to the left or right of the display screen, the image becomes distorted (one side is taller than the other). The *HorizontalKeystone* state variable is used to compensate for this type of distortion.

Note: The following descriptions illustrate the effect of the *HorizontalKeystone* state variable when applied to a projection device that is properly centered with respect to the viewing screen (for example, no misalignment distortion exists).

The left side of the display's output decreases in size as the value becomes more negative (that is: moves farther away from zero in a negative direction). This produces a trapezoidal-shape image with the left and right edges remaining parallel, but with the left side shorter than the right side.

The right side of the display's output decreases in size as the value increases (that is: moves farther from zero in a positive direction). This produces a trapezoidal-shape image with the left and right edge remaining parallel, but with the right side shorter than the left side.

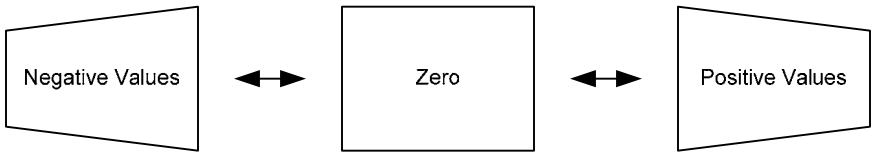

Figure 1: Horizontal Keystone

2.2.14 *VerticalKeystone*

This signed integer state variable represents the current level of compensation for vertical distortion (described below) of the associated display device. Its value ranges from device-specific negative number to a device specific positive number. Zero does not need to be in the middle of this range, although it will be for most devices. A numerical change of 1 corresponds to the smallest incremental change that is supported by the device.

Vertical distortion can occur when the display device is vertical misaligned from the center of the viewing screen. For example, when a video/still-image projection device is moved above or below the display screen, the image becomes distorted (that is: the top and bottom edges have different lengths). The *VerticalKeystone* state variable is used to compensate for this type of distortion.

Note: The following descriptions illustrate the effect of the *VerticalKeystone* state variable when applied to a projection device that is properly centered with respect to the viewing screen (for example, no misalignment distortion exists).

The bottom edge of the display's output decreases in size as the value becomes more negative (that is: moves farther away from zero). This produces a trapezoidal-shape image with the top and bottom edges remaining parallel, but with the bottom edge shorter than the top edge.

The top edge of the display's output decreases in size as the value increases (that is: moves farther from zero in a positive direction). This produces a trapezoidal-shape image with the top and bottom edges remaining parallel, but with the top edge shorter than the bottom edge.

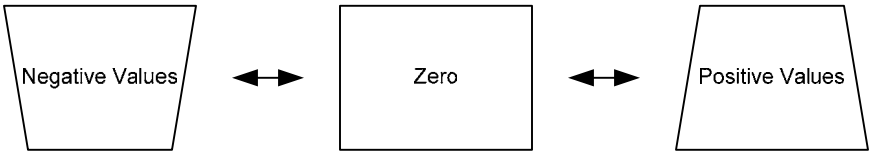

Figure 2: Vertical Keystone

2.2.15 *Mute*

This boolean state variable represents the current mute setting of the associated audio channel. A value of TRUE (that is: a numerical value of 1) indicates that the output of the associated audio channel is currently muted (that is: that channel is not producing any sound).

2.2.16 *Volume*

This unsigned integer state variable represents the current volume setting of the associated audio channel. Its value ranges from a minimum of 0 to some device specific maximum, N. This implies that the device supports exactly (N+1) discrete volume settings for this audio channel. A numerical change of +1 or -1 selects the next available volume setting up or down respectively.

Lower values produce a quieter sound and higher values produce a louder sound. A value of 0 represents the quietest level of sound and a value of N represents the loudest level of sound supported by the device for that channel.

2.2.17 *VolumeDB*

This signed integer state variable represents the current volume setting of the associated audio channel. Its value represents the current volume setting, expressed in decibel (dB). However, to represent volume settings with a resolution better than 1 dB, the integer value MUST be interpreted as having the decimal point between the Most Significant Byte (MSB) and the Least Significant Byte (LSB). This effectively results in a volume setting range from +127.9961 dB (0x7FFF) down to -127.9961 dB (0x8001) in steps of 1/256 dB or 0.00390625 dB (0x0001), providing an available range and resolution far beyond what is typically needed. The value corresponding to 0x8000 is invalid.

Each implementation of this service MUST specify a vendor-defined minimum value (*MinValue*) and maximum value (*MaxValue*). The *MinValue* and *MaxValue* values can be retrieved through the *GetVolumeDBRange()* action. Most implementations will not support all incremental values between the supported minimum and maximum values. If a control point attempts to set *VolumeDB* to an unsupported value, the device will set the volume to the closest setting that is supported by the device for the associated channel. Lower values (including negative values) produce a quieter sound and larger values produce a louder sound. A value of *MinValue* dB represents the quietest level of sound and a value of *MaxValue* dB represents the loudest level of sound supported by the device for that channel.

Note: Since *MinValue* and *MaxValue* will typically be used to generate user interface elements (like volume sliders), it is important to report a reasonable value for *MinValue* rather than the actual value implemented by the physical volume control. For instance, if a physical volume control has a lowest position 0 that is effectively a mute of the output signal (-∞ dB), and the next higher position 1 corresponds to -70 dB, it would not be wise to report *MinValue* = -127.9961 dB (the lowest available value to represent -∞) since this would have adverse results when used to draw a linear dB scale in the user interface. Instead, a more reasonable value of, say -75 dB could be reported to represent the quietest position 0 of the control.

2.2.18 *Loudness*

This Boolean state variable represents the current loudness setting of the associated audio channel. A value of TRUE (that is: a numerical value of 1) indicates that the loudness effect is active.

2.2.19 *A_ARG_TYPE_Channel*

This state variable is introduced to provide type information for the *Channel* input argument in various actions of the RenderingControl service. It is used to identify a particular channel of an audio stream. A channel, except the *Master* channel, is associated with the location of the speaker where the audio data stream is to be presented. It is customary to refer to a channel using the spatial position of the associated speaker as described below.

The *Master* channel is a logical channel and, therefore, has no spatial position associated with it. A one-channel channel cluster does not have spatial position associated with it either and will use the *Master* channel to control its properties.

The following channel spatial positions are defined:

- Master (Master)
- Left Front (LF)
- Right Front (RF)
- Center Front (CF)
- Low Frequency Enhancement (LFE) [Super woofer]
- Left Surround (LS)
- Right Surround (RS)
- Left of Center (LFC) [in front]
- Right of Center (RFC) [in front]
- Surround (SD) [rear]
- Side Left (SL) [left wall]
- Side Right (SR) [right wall]
- Top (T) [overhead]
- Bottom (B) [bottom]

A channel cluster is the collection of all channels, including the *Master* channel, within an audio stream. A single channel (mono) cluster has only one channel – the *Master* channel. A two-channel (stereo) cluster has three channels – the *Master* channel, the Left Front channel, and the Right Front channel. In this specification, only the Master channel is REQUIRED. All other channels are OPTIONAL, see Table 2-16, "allowedValueList for *A_ARG_TYPE_Channel*" for details.

2.2.20 *A_ARG_TYPE_InstanceID*

This state variable is introduced to provide type information for the *InstanceID* input argument in various actions of the RenderingControl service. It specifies the virtual instance of the RenderingControl service to which the associated action MUST be applied. A value of "0" indicates that the action MUST be applied to the global (post-mix) stream, as shown in Figure 4: Virtual Instances of RCS.

2.2.21 *A_ARG_TYPE_PresetName*

This state variable is introduced to provide type information for the *PresetName* input argument in the *SelectPreset* action of the RenderingControl service. This string argument is used to specify the name of a device preset. This MAY include any of the names listed in the *PresetNameList* state variable or any of the predefined presets (listed below) that are supported by the device.

Table 2-18: Predefined Names of Some Common Presets

Value	Definition
"*FactoryDefaults*"	The factory settings defined by the device's manufacturer.

Value	Definition
"*InstallationDefaults*"	The installation settings defined by the installer of the device. This may or may not be the same as the Factory defaults.

2.2.22 *A_ARG_TYPE_DeviceUDN*

This state variable is introduced to provide type information for the *RenderingControlUDN* argument in certain actions. It is a **string** value containing the UDN of the device.

2.2.23 *A_ARG_TYPE_ServiceType*

This state variable is introduced to provide type information for the *ServiceType* argument in certain actions. It is a **string** value containing the service type and version number of a service such as "RenderingControl:2".

2.2.24 *A_ARG_TYPE_ServiceID*

This state variable is introduced to provide type information for the *ServiceId* argument in certain actions. It is a **string** value containing the service ID of a service.

2.2.25 *A_ARG_TYPE_StateVariableValuePairs*

This state variable is introduced to provide type information for the *StateVariableValuePairs* argument in certain actions. This state variable contains a list of state variable names and their values. The list of state variables whose name/value pair is requested is given by another argument to the action. The structure of the *StateVariableValuePairs* argument is defined in [AVS-XSD]. The following XML fragment illustrates a typical example of the schema:

```
<?xml version="1.0" encoding="UTF-8"?>
<stateVariableValuePairs
 xmlns="urn:schemas-upnp-org:av:avs"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="
   urn:schemas-upnp-org:av:avs
    http://www.upnp.org/schemas/av/avs-v1-20060531.xsd">
   <stateVariable variableName="Brightness">
      4
   </stateVariable>
   <stateVariable variableName="Volume" channel="Master">
      50
   </stateVariable>
   <!—- More state variable value pairs can be inserted here -->
</stateVariableValuePairs>
```

The relevant variableNames MUST be either all or a subset (as required) of the defined RenderingControl state variables except for *LastChange*, *PresetNameList*, and any *A_ARG_TYPE_xxx* state variables.

2.2.26 *A_ARG_TYPE_StateVariableList*

This state variable is introduced to provide type information for the *StateVariableList* argument in certain actions. It is a CSV list of state variable names. This variable MAY contain one or more (as required) of the defined RenderingControl state variable names except *LastChange*, *PresetNameList*, and any *A_ARG_TYPE_xxx* state variable names. The asterisk ("*") can be specified to indicate all relevant variable names (excluding *LastChange*, *PresetNameList*, and any *A_ARG_TYPE_xxx* state variables.)

2.2.27 Relationships between State Variables

Except for the *LastChange* and two volume-related state variables (that is: *Volume* and *VolumeDB*), all state variables operate independently. However, whenever any (non A_ARG_TYPE_xxx) state variable changes, a state change descriptor is added to the *LastChange* state variable to reflect the modified state. Refer to the description of the LastChange state variable and overall eventing model for more details.

2.2.27.1 *xxxVideoGain* and *xxxVideoBlackLevel*

As described in Section 2.2.6, "*RedVideoGain*" thru Section 2.2.11, "*BlueVideoBlackLevel*", a pair of state variables is defined to control the output intensity of each primary color (that is: red, green, and blue). The state variable pair associated with each color includes one state variable to control the gain of that color and one state variable to control the minimum output intensity of that color (for example, *RedVideoGain* and *RedVideoBlackLevel*).

Although these two state variables are associated with the same color, they function independently from each other (that is: changing the value of one state variable does not affect the value of the other). However, each state variable within a given pair (that is: associated with a given color) may affect the output intensity of that color. For example, while displaying a static image, increasing the value of either the *RedVideoGain* or *RedVideoBlackLevel* state variables may cause an increase the amount of red that is displayed. Similarly, decreasing either state variable may cause a decrease in the amount of red.

The precise effect of these two variables may vary from device to device. However, as a common example, the *RedVideoGain* controls the multiplication factor between the input and output intensity of the color red and the *RedVideoBlackLevel* controls minimum output intensity of red regardless of the input intensity. Thus, a typical implementation may be described using a variation of the following formula:

Red Output Intensity = *RedVideoGain* * Red Input Intensity + *RedVideoBlackLevel*

2.2.27.2 *Volume* and *VolumeDB*

There MUST be a one-to-one correspondence between the supported values for the *Volume* state variable and the *VolumeDB* state variable as both state variables actually represent the same physical volume control. Therefore, if the *Volume* state variable supports (N+1) values, then the *VolumeDB* state variable MUST also support (N+1) values. In particular, a *Volume* value of 0 MUST correspond to a *VolumeDB* value of *MinValue* dB, and a *Volume* value of N MUST correspond to a *VolumeDB* value of *MaxValue* dB. Note that although the *Volume* state variable can take all (N+1) contiguous integer values between 0 and N, this does not imply that the actual volume increments MUST be constant over the entire supported range. The *Volume* state variable MUST be considered merely as an index into an array Vol() of possible volume settings that the physical volume control actually implements. Likewise, the *VolumeDB* state variable can only take the (N+1) discrete values contained in the above mentioned array.

VolumeDB = Vol(*Volume*), *Volume* Є [0,N]

As an example, consider a physical volume control that implements 45 different positions (N=44). The loudest position corresponds to 0 dB (*MaxValue* = 0 dB) and the quietest position corresponds to -72 dB (*MinValue* = -72 dB). The control has 3 zones that each offer different step resolutions: Between 0 and -24 dB, the resolution is 1 dB, between -24 and -48 dB, the resolution is 2 dB, and between -48 and -72 dB, the resolution is 3 dB. The following figure shows the relationship between the *Volume* and *VolumeDB* state variables.

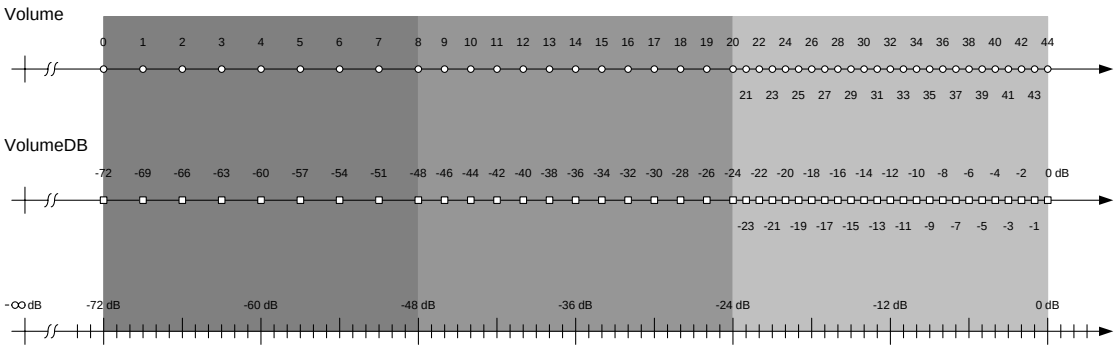

Figure 3: Relationship between *Volume* and *VolumeDB*

2.3 Eventing and Moderation

As the table below summarizes, the RenderingControl specification uses moderated eventing for only one of its standard state variables and no eventing for the rest.

Table 2-19: Event moderation

Variable Name	Evented	Moderated Event	Max Event Rate[1]	Logical Combination	Min Delta per Event[2]
LastChange	*YES*	*YES*	*0.2 seconds*		
PresetNameList	*NO*	*NO*			
Brightness	*NO*	*NO*			
Contrast	*NO*	*NO*			
Sharpness	*NO*	*NO*			
RedVideoGain	*NO*	*NO*			
GreenVideoGain	*NO*	*NO*			
BlueVideoGain	*NO*	*NO*			
RedVideoBlackLevel	*NO*	*NO*			
GreenVideoBlackLevel	*NO*	*NO*			
BlueVideoBlackLevel	*NO*	*NO*			
ColorTemperature	*NO*	*NO*			
HorizontalKeystone	*NO*	*NO*			
VerticalKeystone	*NO*	*NO*			
Mute	*NO*	*NO*			
Volume	*NO*	*NO*			
VolumeDB	*NO*	*NO*			
Loudness	*NO*	*NO*			
A_ARG_TYPE_Channel	*NO*	*NO*			
A_ARG_TYPE_InstanceID	*NO*	*NO*			
A_ARG_TYPE_PresetName	*NO*	*NO*			

Variable Name	Evented	Moderated Event	Max Event Rate[1]	Logical Combination	Min Delta per Event[2]
A_ARG_TYPE_DeviceUDN	*NO*	*NO*			
A_ARG_TYPE_ServiceType	*NO*	*NO*			
A_ARG_TYPE_ServiceID	*NO*	*NO*			
A_ARG_TYPE_ StateVariableValuePairs	*NO*	*NO*			
A_ARG_TYPE_ StateVariableList	*NO*	*NO*			
Non-standard state variables implemented by a UPnP vendor go here	*NO*	*NO*			

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

[2] (N) * (allowedValueRange Step).

Note: Non-standard state variables MUST also be evented through the *LastChange* event mechanism.

2.3.1 Event Model

Since the RenderingControl service supports multiple virtual instances (via the *InstanceID* argument included in each action), the traditional UPnP eventing model is unable to differentiate between multiple instances of the same state variable. Therefore, the RenderingControl service event model defines a specialized state variable (*LastChange*) that is used exclusively for eventing individual state changes. In this model, the *LastChange* state change is the only variable that is evented using the standard UPnP event mechanism. All other state variables are indirectly evented via the *LastChange* state variable. (Note: A_ARG_TYPE_ state variables are not evented, either directly or indirectly.)

When the value of a state variable changes, information about that change is added to the *LastChange* state variable as described in Section 2.2.1, "*LastChange*." As a result of modifying the *LastChange* state variable, its new value (that is: the information describing the original state change) is evented using the standard UPnP eventing mechanism. In this manner, the change to the original state variable is indirectly evented to all interested control points.

Since the *LastChange* state variable is a moderated state variable, multiple state changes are accumulated in the *LastChange* state variable until its moderation period expires. When this occurs, the current value of the *LastChange* state variable is sent out using the standard UPnP eventing mechanism. This notification informs interested control points of all state changes that have occurred since the previous *LastChange* event was sent.

After the *LastChange* state variable is evented, its contents is cleared out in preparation for the next state change. (Note: The act of clearing out the *stale* contents of the *LastChange* state variable does not need to generate another event notification even though its value has changed. Doing so would only generate unnecessary network traffic.). Because the *LastChange* state variable is moderated, a given state variable MAY change multiple times before the current moderation period expires. In such cases, the *LastChange* state variable will contain a single entry for that state variable reflecting its current (most recent) value.

The standard UPnP event mechanism indicates that when a control point subscribes to receive events, the current values of all evented state variables are returned to the subscriber. However, since the *LastChange* state variable is the only state variable that is directly evented (that is: all other state variables are indirectly evented via the *LastChange* state variable), it is not very meaningful to respond to an event subscription request with the current value of the *LastChange* state variable. Its value is too transitory to be of any use to the subscribing control point. Therefore, when an event subscription is received, the device MUST respond

with the current values of all (indirectly evented) state variables within all valid instances of this service. Refer to Section 2.2.1, "*LastChange*" for additional information.

2.4 Actions

The following tables and subsections define the various RenderingControl actions. Except where noted, if an invoked action returns an error, the state of the device will be unaffected.

Table 2-20: Actions

Name	R/O[11]
ListPresets()	*R*
SelectPreset()	*R*
GetBrightness()	*O*
SetBrightness()	*O*
GetContrast()	*O*
SetContrast()	*O*
GetSharpness()	*O*
SetSharpness()	*O*
GetRedVideoGain()	*O*
SetRedVideoGain()	*O*
GetGreenVideoGain()	*O*
SetGreenVideoGain()	*O*
GetBlueVideoGain()	*O*
SetBlueVideoGain()	*O*
GetRedVideoBlackLevel()	*O*
SetRedVideoBlackLevel()	*O*
GetGreenVideoBlackLevel()	*O*
SetGreenVideoBlackLevel()	*O*
GetBlueVideoBlackLevel()	*O*
SetBlueVideoBlackLevel()	*O*
GetColorTemperature()	*O*
SetColorTemperature()	*O*
GetHorizontalKeystone()	*O*
SetHorizontalKeystone()	*O*
GetVerticalKeystone()	*O*
SetVerticalKeystone()	*O*
GetMute()	*O*
SetMute()	*O*
GetVolume()	*O*

Name	R/O[11]
SetVolume()	*O*
GetVolumeDB()	*O*
SetVolumeDB()	*O*
GetVolumeDBRange()	*O*
GetLoudness()	*O*
SetLoudness()	*O*
GetStateVariables()	*O*
SetStateVariables()	*O*
Non-standard actions implemented by a UPnP vendor go here	*X*

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

Note: Non-standard actions MUST be implemented in such a way that they do not interfere with the basic operation of the RenderingControl service; that is: these actions MUST be optional and do not need to be invoked for the RenderingControl service to operate normally.

2.4.1 *ListPresets()*

This action returns a list of the currently defined presets. The *CurrentPresetNameList* OUT argument contains a comma-separated list of preset names that include both predefined (static) presets and user-defined (dynamically created) presets that may be created via a private vendor-defined action.

2.4.1.1 Arguments

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

Argument	Direction	relatedStateVariable
InstanceID	*IN*	*A_ARG_TYPE_InstanceID*
CurrentPresetNameList	*OUT*	*PresetNameList*

2.4.1.2 Dependency on State

None.

2.4.1.3 Effect on State

None.

2.4.1.4 Errors

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

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.
702	Invalid InstanceID	The specified *InstanceID* is invalid.

2.4.2 *SelectPreset()*

This action restores (a subset) of the state variables to the values associated with the specified preset. The specified preset name MAY be one of the predefined presets (listed in Table 2-18, "Predefined Names of Some Common Presets") or one of the user-defined presets that may have been created via a private vendor-defined action. The selected preset determines which state variables will be affected.

2.4.2.1 Arguments

Table 2-23: Arguments for *SelectPreset()*

Argument	Direction	relatedStateVariable
InstanceID	*IN*	*A_ARG_TYPE_InstanceID*
PresetName	*IN*	*A_ARG_TYPE_PresetName*

2.4.2.2 Dependency on State

None.

2.4.2.3 Effect on State

This action sets the state of the service to the values associated with the specified preset.

2.4.2.4 Errors

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

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	Invalid Name	The specified name is not a valid preset name.
702	Invalid InstanceID	The specified *InstanceID* is invalid.

2.4.3 *GetBrightness()*

This action retrieves the current value of the *Brightness* state variable of the specified instance of this service.

2.4.3.1 Arguments

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

Argument	Direction	relatedStateVariable
InstanceID	*IN*	*A_ARG_TYPE_InstanceID*
CurrentBrightness	*OUT*	*Brightness*

2.4.3.2 Dependency on State

None.

2.4.3.3 Effect on State

None.

2.4.3.4 Errors

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

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.
702	Invalid InstanceID	The specified *InstanceID* is invalid.

2.4.4 *SetBrightness()*

This action sets the *Brightness* state variable of the specified instance of this service to the specified value.

2.4.4.1 Arguments

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

Argument	Direction	relatedStateVariable
InstanceID	IN	A_ARG_TYPE_InstanceID
DesiredBrightness	IN	Brightness

2.4.4.2 Dependency on State

None.

2.4.4.3 Effect on State

This action affects the *Brightness* state variable of the specified instance of this service.

2.4.4.4 Errors

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

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.
702	Invalid InstanceID	The specified *InstanceID* is invalid.

2.4.5 *GetContrast()*

This action retrieves the current value of the *Contrast* state variable of the specified instance of this service.

2.4.5.1 Arguments

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

Argument	Direction	relatedStateVariable
InstanceID	*IN*	*A_ARG_TYPE_InstanceID*
CurrentContrast	*OUT*	*Contrast*

2.4.5.2 Dependency on State

None.

2.4.5.3 Effect on State

None.

2.4.5.4 Errors

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

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.
702	Invalid InstanceID	The specified *InstanceID* is invalid.

2.4.6 *SetContrast()*

This action sets the *Contrast* state variable of the specified instance of this service to the specified value.

2.4.6.1 Arguments

Table 2-31: Arguments for *SetContrast()*

Argument	Direction	relatedStateVariable
InstanceID	*IN*	*A_ARG_TYPE_InstanceID*
DesiredContrast	*IN*	*Contrast*

2.4.6.2 Dependency on State

None.

2.4.6.3 Effect on State

This action affects the *Contrast* state variable of the specified instance of this service.

2.4.6.4 Errors

Table 2-32: Error Codes for *SetContrast()*

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.
702	Invalid InstanceID	The specified *InstanceID* is invalid.

2.4.7 *GetSharpness()*

This action retrieves the current value of the *Sharpness* state variable of the specified instance of this service.

2.4.7.1 Arguments

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

Argument	Direction	relatedStateVariable
InstanceID	*IN*	*A_ARG_TYPE_InstanceID*
CurrentSharpness	*OUT*	*Sharpness*

2.4.7.2 Dependency on State

None.

2.4.7.3 Effect on State

None.

2.4.7.4 Errors

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

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.
702	Invalid InstanceID	The specified *InstanceID* is invalid.

2.4.8 *SetSharpness()*

This action sets the *Sharpness* state variable of the specified instance of this service to the specified value.

2.4.8.1 Arguments

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

Argument	Direction	relatedStateVariable
InstanceID	*IN*	*A_ARG_TYPE_InstanceID*
DesiredSharpness	*IN*	*Sharpness*

2.4.8.2 Dependency on State

None.

2.4.8.3 Effect on State

This action affects the *Sharpness* state variable of the specified instance of this service.

2.4.8.4 Errors

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

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

errorCode	errorDescription	Description
600-699	TBD	See UPnP Device Architecture section on Control.
702	Invalid InstanceID	The specified *InstanceID* is invalid.

2.4.9 *GetRedVideoGain()*

This action retrieves the current value of the *RedVideoGain* state variable of the specified instance of this service.

2.4.9.1 Arguments

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

Argument	Direction	relatedStateVariable
InstanceID	*IN*	*A_ARG_TYPE_InstanceID*
CurrentRedVideoGain	*OUT*	*RedVideoGain*

2.4.9.2 Dependency on State

None.

2.4.9.3 Effect on State

None.

2.4.9.4 Errors

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

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.
702	Invalid InstanceID	The specified *InstanceID* is invalid.

2.4.10 *SetRedVideoGain()*

This action sets the *RedVideoGain* state variable of the specified instance of this service to the specified value.

2.4.10.1 Arguments

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

Argument	Direction	relatedStateVariable
InstanceID	*IN*	*A_ARG_TYPE_InstanceID*
DesiredRedVideoGain	*IN*	*RedVideoGain*

2.4.10.2 Dependency on State

None.

2.4.10.3 Effect on State

This action affects the *RedVideoGain* state variable of the specified instance of this service.

2.4.10.4 Errors

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

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.
702	Invalid InstanceID	The specified *InstanceID* is invalid.

2.4.11 *GetGreenVideoGain()*

This action retrieves the current value of the *GreenVideoGain* state variable of the specified instance of this service.

2.4.11.1 Arguments

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

Argument	Direction	relatedStateVariable
InstanceID	*IN*	*A_ARG_TYPE_InstanceID*
CurrentGreenVideoGain	*OUT*	*GreenVideoGain*

2.4.11.2 Dependency on State

None.

2.4.11.3 Effect on State

None.

2.4.11.4 Errors

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

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.
702	Invalid InstanceID	The specified *InstanceID* is invalid.

2.4.12 *SetGreenVideoGain()*

This action sets the *GreenVideoGain* state variable of the specified instance of this service to the specified value.

2.4.12.1 Arguments

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

Argument	Direction	relatedStateVariable
InstanceID	*IN*	*A_ARG_TYPE_InstanceID*
DesiredGreenVideoGain	*IN*	*GreenVideoGain*

2.4.12.2 Dependency on State

None.

2.4.12.3 Effect on State

This action affects the *GreenVideoGain* state variable of the specified instance of this service.

2.4.12.4 Errors

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

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.
702	Invalid InstanceID	The specified *InstanceID* is invalid.

2.4.13 *GetBlueVideoGain()*

This action retrieves the current value of the *BlueVideoGain* state variable of the specified instance of this service.

2.4.13.1 Arguments

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

Argument	Direction	relatedStateVariable
InstanceID	*IN*	*A_ARG_TYPE_InstanceID*
CurrentBlueVideoGain	*OUT*	*BlueVideoGain*

2.4.13.2 Dependency on State

None.

2.4.13.3 Effect on State

None.

2.4.13.4 Errors

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

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

errorCode	errorDescription	Description
600-699	TBD	See UPnP Device Architecture section on Control.
702	Invalid InstanceID	The specified *InstanceID* is invalid.

2.4.14 *SetBlueVideoGain()*

This action sets the *BlueVideoGain* state variable of the specified instance of this service to the specified value.

2.4.14.1 Arguments

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

Argument	Direction	relatedStateVariable
InstanceID	*IN*	*A_ARG_TYPE_InstanceID*
DesiredBlueVideoGain	*IN*	*BlueVideoGain*

2.4.14.2 Dependency on State

None.

2.4.14.3 Effect on State

This action affects the *BlueVideoGain* state variable of the specified instance of this service.

2.4.14.4 Errors

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

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.
702	Invalid InstanceID	The specified *InstanceID* is invalid.

2.4.15 *GetRedVideoBlackLevel()*

This action retrieves the current value of the *RedVideoBlackLevel* state variable of the specified instance of this service.

2.4.15.1 Arguments

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

Argument	Direction	relatedStateVariable
InstanceID	*IN*	*A_ARG_TYPE_InstanceID*
CurrentRedVideoBlackLevel	*OUT*	*RedVideoBlackLevel*

2.4.15.2 Dependency on State

None.

2.4.15.3 Effect on State

None.

2.4.15.4 Errors

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

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.
702	Invalid InstanceID	The specified *InstanceID* is invalid.

2.4.16 *SetRedVideoBlackLevel()*

This action sets the *RedVideoBlackLevel* state variable of the specified instance of this service to the specified value.

2.4.16.1 Arguments

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

Argument	Direction	relatedStateVariable
InstanceID	*IN*	*A_ARG_TYPE_InstanceID*
DesiredRedVideoBlackLevel	*IN*	*RedVideoBlackLevel*

2.4.16.2 Dependency on State

None.

2.4.16.3 Effect on State

This action affects the *RedVideoBlackLevel* state variable of the specified instance of this service.

2.4.16.4 Errors

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

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.
702	Invalid InstanceID	The specified *InstanceID* is invalid.

2.4.17 *GetGreenVideoBlackLevel()*

This action retrieves the current value of the *GreenVideoBlackLevel* state variable of the specified instance of this service.

2.4.17.1 Arguments

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

Argument	Direction	relatedStateVariable
InstanceID	*IN*	*A_ARG_TYPE_InstanceID*
CurrentGreenVideoBlackLevel	*OUT*	*GreenVideoBlackLevel*

2.4.17.2 Dependency on State

None.

2.4.17.3 Effect on State

None.

2.4.17.4 Errors

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

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.
702	Invalid InstanceID	The specified *InstanceID* is invalid.

2.4.18 *SetGreenVideoBlackLevel()*

This action sets the *GreenVideoBlackLevel* state variable of the specified instance of this service to the specified value.

2.4.18.1 Arguments

Table 2-55: Arguments for *SetGreenVideoBlackLevel()*

Argument	Direction	relatedStateVariable
InstanceID	*IN*	*A_ARG_TYPE_InstanceID*
DesiredGreenVideoBlackLevel	*IN*	*GreenVideoBlackLevel*

2.4.18.2 Dependency on State

None.

2.4.18.3 Effect on State

This action affects the *GreenVideoBlackLevel* state variable.

2.4.18.4 Errors

Table 2-56: Error Codes for *SetGreenVideoBlackLevel()*

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

errorCode	errorDescription	Description
600-699	TBD	See UPnP Device Architecture section on Control.
702	Invalid InstanceID	The specified *InstanceID* is invalid.

2.4.19 *GetBlueVideoBlackLevel()*

This action retrieves the current value of the *BlueVideoBlackLevel* state variable of the specified instance of this service.

2.4.19.1 Arguments

Table 2-57: Arguments for *GetBlueVideoBlackLevel()*

Argument	Direction	relatedStateVariable
InstanceID	*IN*	*A_ARG_TYPE_InstanceID*
CurrentBlueVideoBlackLevel	*OUT*	*BlueVideoBlackLevel*

2.4.19.2 Dependency on State

None.

2.4.19.3 Effect on State

None.

2.4.19.4 Errors

Table 2-58: Error Codes for *GetBlueVideoBlackLevel()*

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.
702	Invalid InstanceID	The specified *InstanceID* is invalid.

2.4.20 *SetBlueVideoBlackLevel()*

This action sets the *BlueVideoBlackLevel* state variable of the specified instance of this service to the specified value.

2.4.20.1 Arguments

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

Argument	Direction	relatedStateVariable
InstanceID	*IN*	*A_ARG_TYPE_InstanceID*
DesiredBlueVideoBlackLevel	*IN*	*BlueVideoBlackLevel*

2.4.20.2 Dependency on State

None.

2.4.20.3 Effect on State

This action affects the *BlueVideoBlackLevel* state variable of the specified instance of this service.

2.4.20.4 Errors

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

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.
702	Invalid InstanceID	The specified *InstanceID* is invalid.

2.4.21 *GetColorTemperature()*

This action retrieves the current value of the *ColorTemperature* state variable of the specified instance of this service.

2.4.21.1 Arguments

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

Argument	Direction	relatedStateVariable
InstanceID	*IN*	*A_ARG_TYPE_InstanceID*
CurrentColorTemperature	*OUT*	*ColorTemperature*

2.4.21.2 Dependency on State

None.

2.4.21.3 Effect on State

None.

2.4.21.4 Errors

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

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.
702	Invalid InstanceID	The specified *InstanceID* is invalid.

2.4.22 *SetColorTemperature()*

This action sets the *ColorTemperature* state variable of the specified instance of this service to the specified value.

2.4.22.1 Arguments

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

Argument	Direction	relatedStateVariable
InstanceID	*IN*	*A_ARG_TYPE_InstanceID*
DesiredColorTemperature	*IN*	*ColorTemperature*

2.4.22.2 Dependency on State

None.

2.4.22.3 Effect on State

This action affects the *ColorTemperature* state variable of the specified instance of this service.

2.4.22.4 Errors

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

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.
702	Invalid InstanceID	The specified *InstanceID* is invalid.

2.4.23 *GetHorizontalKeystone()*

This action retrieves the current value of the *HorizontalKeystone* state variable of the specified instance of this service.

2.4.23.1 Arguments

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

Argument	Direction	relatedStateVariable
InstanceID	*IN*	*A_ARG_TYPE_InstanceID*
CurrentHorizontalKeystone	*OUT*	*HorizontalKeystone*

2.4.23.2 Dependency on State

None.

2.4.23.3 Effect on State

None.

2.4.23.4 Errors

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

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

errorCode	errorDescription	Description
600-699	TBD	See UPnP Device Architecture section on Control.
702	Invalid InstanceID	The specified *InstanceID* is invalid.

2.4.24 *SetHorizontalKeystone()*

This action sets the *HorizontalKeystone* state variable of the specified instance of this service to the specified value.

2.4.24.1 Arguments

Table 2-67: Arguments for *SetHorizontalKeystone()*

Argument	Direction	relatedStateVariable
InstanceID	*IN*	*A_ARG_TYPE_InstanceID*
DesiredHorizontalKeystone	*IN*	*HorizontalKeystone*

2.4.24.2 Dependency on State

None.

2.4.24.3 Effect on State

This action affects the *HorizontalKeystone* state variable of the specified instance of this service.

2.4.24.4 Errors

Table 2-68: Error Codes for *SetHorizontalKeystone()*

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.
702	Invalid InstanceID	The specified *InstanceID* is invalid.

2.4.25 *GetVerticalKeystone()*

This action retrieves the current value of the *VerticalKeystone* state variable of the specified instance of this service.

2.4.25.1 Arguments

Table 2-69: Arguments for *GetVerticalKeystone()*

Argument	Direction	relatedStateVariable
InstanceID	*IN*	*A_ARG_TYPE_InstanceID*
CurrentVerticalKeystone	*OUT*	*VerticalKeystone*

2.4.25.2 Dependency on State

None.

2.4.25.3 Effect on State

None.

2.4.25.4 Errors

Table 2-70: Error Codes for *GetVerticalKeystone()*

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.
702	Invalid InstanceID	The specified *InstanceID* is invalid.

2.4.26 *SetVerticalKeystone()*

This action sets the *VerticalKeystone* state variable of the specified instance of this service to the specified value.

2.4.26.1 Arguments

Table 2-71: Arguments for *SetVerticalKeystone()*

Argument	Direction	relatedStateVariable
InstanceID	*IN*	*A_ARG_TYPE_InstanceID*
DesiredVerticalKeystone	*IN*	*VerticalKeystone*

2.4.26.2 Dependency on State

None.

2.4.26.3 Effect on State

This action affects the *VerticalKeystone* state variable of the specified instance of this service.

2.4.26.4 Errors

Table 2-72: Error Codes for *SetVerticalKeystone()*

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.
702	Invalid InstanceID	The specified *InstanceID* is invalid.

2.4.27 *GetMute()*

This action retrieves the current value of the *Mute* setting of the channel for the specified instance of this service.

2.4.27.1 Arguments

Table 2-73: Arguments for *GetMute()*

Argument	Direction	relatedStateVariable
InstanceID	*IN*	*A_ARG_TYPE_InstanceID*
Channel	*IN*	*A_ARG_TYPE_Channel*
CurrentMute	*OUT*	*Mute*

2.4.27.2 Dependency on State

None.

2.4.27.3 Effect on State

None.

2.4.27.4 Errors

Table 2-74: Error Codes for *GetMute()*

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.
702	Invalid InstanceID	The specified *InstanceID* is invalid.
703	Invalid Channel	The specified *Channel* is invalid.

2.4.28 *SetMute()*

This action sets the *Mute* state variable of the specified instance of this service to the specified value.

2.4.28.1 Arguments

Table 2-75: Arguments for *SetMute()*

Argument	Direction	relatedStateVariable
InstanceID	*IN*	*A_ARG_TYPE_InstanceID*
Channel	*IN*	*A_ARG_TYPE_Channel*
DesiredMute	*IN*	*Mute*

2.4.28.2 Dependency on State

None.

2.4.28.3 Effect on State

This action affects the *Mute* state variable of the specified instance of this service.

2.4.28.4 Errors

Table 2-76: Error Codes for *SetMute()*

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.
702	Invalid InstanceID	The specified *InstanceID* is invalid.
703	Invalid Channel	The specified *Channel* is invalid.

2.4.29 *GetVolume()*

This action retrieves the current value of the *Volume* state variable of the specified channel for the specified instance of this service. The *CurrentVolume* (OUT) argument contains a value ranging from 0 to a device-specific maximum, N. See Section 2.2.16, "*Volume*" for more details.

2.4.29.1 Arguments

Table 2-77: Arguments for *GetVolume()*

Argument	Direction	relatedStateVariable
InstanceID	*IN*	*A_ARG_TYPE_InstanceID*
Channel	*IN*	*A_ARG_TYPE_Channel*
CurrentVolume	*OUT*	*Volume*

2.4.29.2 Dependency on State

None.

2.4.29.3 Effect on State

None.

2.4.29.4 Errors

Table 2-78: Error Codes for *GetVolume()*

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.
702	Invalid InstanceID	The specified *InstanceID* is invalid.
703	Invalid Channel	The specified *Channel* is invalid.

2.4.30 *SetVolume()*

This action sets the *Volume* state variable of the specified instance and channel to the specified value. The *DesiredVolume* input argument contains a value ranging from 0 to a device-specific maximum, N. See Section 2.2.16, "*Volume*" for more details.

2.4.30.1 Arguments

Table 2-79: Arguments for *SetVolume()*

Argument	Direction	relatedStateVariable
InstanceID	*IN*	*A_ARG_TYPE_InstanceID*
Channel	*IN*	*A_ARG_TYPE_Channel*
DesiredVolume	*IN*	*Volume*

2.4.30.2 Dependency on State

None.

2.4.30.3 Effect on State

This action affects the *Volume* and *VolumeDB* state variables of the specified instance and channel. Both state variables need to change consistently.

2.4.30.4 Errors

Table 2-80: Error Codes for *SetVolume()*

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.
702	Invalid InstanceID	The specified *InstanceID* is invalid.
703	Invalid Channel	The specified *Channel* is invalid.

2.4.31 *GetVolumeDB()*

This action retrieves the current value of the *VolumeDB* state variable of the channel for the specified instance of this service. The *CurrentVolume* (OUT) argument represents the current volume setting in units of 1/256 decibels (dB). See Section 2.2.17, "*VolumeDB*" for more details.

2.4.31.1 Arguments

Table 2-81: Arguments for *GetVolumeDB()*

Argument	Direction	relatedStateVariable
InstanceID	*IN*	*A_ARG_TYPE_InstanceID*
Channel	*IN*	*A_ARG_TYPE_Channel*
CurrentVolume	*OUT*	*VolumeDB*

2.4.31.2 Dependency on State

None.

2.4.31.3 Effect on State

None.

2.4.31.4 Errors

Table 2-82: Error Codes for *GetVolumeDB()*

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.
702	Invalid InstanceID	The specified *InstanceID* is invalid.
703	Invalid Channel	The specified *Channel* is invalid.

2.4.32 *SetVolumeDB()*

This action sets the *VolumeDB* state variable of the specified instance and channel to the specified value. The *DesiredVolume* argument represents the desired volume setting in units of 1/256 decibels (dB). See Section 2.2.17, "*VolumeDB*" for more details.

2.4.32.1 Arguments

Table 2-83: Arguments for *SetVolumeDB()*

Argument	Direction	relatedStateVariable
InstanceID	*IN*	*A_ARG_TYPE_InstanceID*
Channel	*IN*	*A_ARG_TYPE_Channel*
DesiredVolume	*IN*	*VolumeDB*

2.4.32.2 Dependency on State

None.

2.4.32.3 Effect on State

This action affects the *Volume* and *VolumeDB* state variables of the specified instance and channel. Both state variables need to change consistently.

2.4.32.4 Errors

Table 2-84: Error Codes for *SetVolumeDB()*

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.
702	Invalid InstanceID	The specified *InstanceID* is invalid.
703	Invalid Channel	The specified *Channel* is invalid.

2.4.33 *GetVolumeDBRange()*

This action retrieves the valid range for the *VolumeDB* state variable of the channel for the specified instance of this service. The *MinValue* and *MaxValue* (OUT) arguments identify the range of valid values

for the *VolumeDB* state variable in units of 1/256 decibels (dB). See Section 2.2.17, "*VolumeDB*" for more details.

2.4.33.1 Arguments

Table 2-85: Arguments for *GetVolumeDBRange()*

Argument	Direction	relatedStateVariable
InstanceID	*IN*	*A_ARG_TYPE_InstanceID*
Channel	*IN*	*A_ARG_TYPE_Channel*
MinValue	*OUT*	*VolumeDB*
MaxValue	*OUT*	*VolumeDB*

2.4.33.2 Dependency on State

None.

2.4.33.3 Effect on State

None.

2.4.33.4 Errors

Table 2-86: Error Codes for *GetVolumeDBRange()*

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.
702	Invalid InstanceID	The specified *InstanceID* is invalid.
703	Invalid Channel	The specified *Channel* is invalid.

2.4.34 *GetLoudness()*

This action retrieves the current value of the *Loudness* setting of the channel for the specified instance of this service.

2.4.34.1 Arguments

Table 2-87: Arguments for *GetLoudness()*

Argument	Direction	relatedStateVariable
InstanceID	*IN*	*A_ARG_TYPE_InstanceID*
Channel	*IN*	*A_ARG_TYPE_Channel*
CurrentLoudness	*OUT*	*Loudness*

2.4.34.2 Dependency on State

None.

2.4.34.3 Effect on State

None.

2.4.34.4 Errors

Table 2-88: Error Codes for *GetLoudness()*

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.
702	Invalid InstanceID	The specified *InstanceID* is invalid.
703	Invalid Channel	The specified *Channel* is invalid.

2.4.35 *SetLoudness()*

This action sets the specified value of the *Loudness* state variable of the channel for the specified instance of this service to the specified value.

2.4.35.1 Arguments

Table 2-89: Arguments for *SetLoudness()*

Argument	Direction	relatedStateVariable
InstanceID	*IN*	*A_ARG_TYPE_InstanceID*
Channel	*IN*	*A_ARG_TYPE_Channel*
DesiredLoudness	*IN*	*Loudness*

2.4.35.2 Dependency on State

None.

2.4.35.3 Effect on State

This action affects the *Loudness* state variable of the specified instance of this service.

2.4.35.4 Errors

Table 2-90: Error Codes for *SetLoudness()*

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.
702	Invalid InstanceID	The specified *InstanceID* is invalid.
703	Invalid Channel	The specified *Channel* is invalid.

2.4.36 *GetStateVariables()*

This action returns the current collection of RenderingControl state variable names and their respective values that are associated with the RenderingControl instance indicated by the input argument *InstanceID*.

The *StateVariableList* argument specifies which state variables are captured. Vendor-extended state variables can be specified in this argument as well. If the value of the *StateVariableList* argument is set to "*", the action MUST return all the supported state variables of the service, including the vendor-extended state variables except for *LastChange*, *PresetNameList*, and any *A_ARG_TYPE_xxx* variables. When the action fails and the error code indicates "invalid StateVariableList", the control point should inspect the list or invoke successive *Getxxx()* actions for each of the state variables instead. RenderingControl service implementations that want to participate in scenarios that use bookmarks MUST implement this optional action. Furthermore, when creating or manipulating bookmarks, control points should set the *StatevariableList* argument to "*" when invoking this action. This ensures that the maximum available set of state information is stored within the bookmark item.

2.4.36.1 Arguments

Table 2-91: Arguments for *GetStateVariables()*

Argument	Direction	relatedStateVariable
InstanceID	*IN*	*A_ARG_TYPE_InstanceID*
StateVariableList	*IN*	*A_ARG_TYPE_StateVariableList*
StateVariableValuePairs	*OUT*	*A_ARG_TYPE_StateVariableValuePairs*

2.4.36.2 Dependency on State

None.

2.4.36.3 Effect on State

None.

2.4.36.4 Errors

Table 2-92: Error Codes for *GetStateVariables()*

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.
702	Invalid InstanceID	The specified *InstanceID* is invalid for this AVTransport.
704	Invalid StateVariableList	Some of the variables are invalid.
705	Ill-formed CSV List	The CSV list is not well formed.

2.4.37 *SetStateVariables()*

This action extracts the values from the *StateVariableValuePairs* IN argument and copies these values to the corresponding RenderingControl state variables associated with the RenderingControl instance indicated by the input argument *InstanceID*. The *RenderingControlUDN*, *ServiceType* and *ServiceId* argument values are used for compatibility checking by the device. If this action is invoked to replace all of the state variable values, the device MUST check whether the *RenderingControlUDN*, *ServiceType* and *ServiceId* input arguments match those of the device. If this is the case, all state variable values will be replaced. Otherwise, the current state variable values MUST NOT be overwritten and the appropriate error code MUST be returned. The *StateVariableList* argument is a CSV list of state variable names that were accepted by the RenderingControl service. RenderingControl service implementations that want to

participate in scenarios that use bookmarks MUST implement this optional action. When the StateVariableValuePairs IN parameter includes variable name/value pairs which cannot be set (such as *LastChange* and *PresetNameList*) the action must return the appropriate error code.

2.4.37.1 Arguments

Table 2-93: Arguments for *SetStateVariables()*

Argument	Direction	relatedStateVariable
InstanceID	*IN*	*A_ARG_TYPE_InstanceID*
RenderingControlUDN	*IN*	*A_ARG_TYPE_DeviceUDN*
ServiceType	*IN*	*A_ARG_TYPE_ServiceType*
ServiceId	*IN*	*A_ARG_TYPE_ServiceID*
StateVariableValuePairs	*IN*	*A_ARG_TYPE_StateVariableValuePairs*
StateVariableList	*OUT*	*A_ARG_TYPE_StateVariableList*

2.4.37.2 Dependency on State

None.

2.4.37.3 Effect on State

None.

2.4.37.4 Errors

Table 2-94: Error Codes for *SetStateVariables()*

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.
702	Invalid InstanceID	The specified *InstanceID* is invalid for this AVTransport.
706	Invalid State Variable Value	One of the StateVariableValuePairs contains an invalid value.
707	Invalid MediaRenderer's UDN	The specified MediaRenderer's UDN is different from the UDN value of the MediaRenderer.
708	Invalid Service Type	The specified *ServiceType* is invalid.
709	Invalid Service Id	The specified *ServiceId* is invalid.
710	State Variables Specified Improperly	*StateVariableValuePairs* includes variables that are not allowed to be set. For example, *LastChange* and/or *PresetNameList* must not be included.

2.4.38 Relationships Between Actions

There is no inherent relationship between any of the various actions. All actions MAY be called in any order.

2.4.39 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 SHOULD be returned.

Table 2-95: 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.
701	Invalid Name	The specified name is not a valid preset name.
702	Invalid InstanceID	The specified *InstanceID* is invalid.
703	Invalid Channel	The specified *Channel* is invalid.
704	Invalid StateVariableList	Some of the variables are invalid.
705	Ill-formed CSV List	The CSV list is not well formed.
706	Invalid State Variable Value	One of the StateVariableValuePairs contains an invalid value.
707	Invalid MediaRenderer's UDN	The specified MediaRenderer's UDN is different from the UDN value of the MediaRenderer.
708	Invalid Service Type	The specified ServiceType is invalid.
709	Invalid Service Id	The specified ServiceId is invalid.
710	State Variables Specified Improperly	StateVariableValuePairs includes variables that are not allowed to be set. For example, LastChange and/or PresetNameList must not be included.

Note 1: The errorDescription field returned by an action does not necessarily contain human-readable text (for example, as indicated in the second column of the Error Code tables.) It may contain machine-readable information that provides more detailed information about the error. It is therefore not advisable for a control point to blindly display the errorDescription field contents to the user.

Note 2: 800-899 Error Codes are not permitted for standard actions. See UPnP Device Architecture section on Control for more details.

2.5 Theory of Operation

2.5.1 Multi-input Devices

Many traditional rendering devices are capable of receiving and rendering only a single item of content at a time. For example, traditional TVs can only receive and display a single TV show at a time, and a stereo system can only play a single song at a time. However, more and more devices are able to receive and render multiple items of content at the same time even though there is only a single piece of rendering hardware (for example, a single TV screen or a single set of speakers). This capability is known as *mixing*.

As an example, while watching TV, a small Picture-in-a-Picture (PIP) can be overlaid on top of the main TV show so that another TV show (or VCR tape) can be watched at the same time. Although the TV contains a single set of output hardware (for example, a single screen), the TV can take multiple items of

content, mix them together, and render them both on the one screen. Similarly, a karaoke system takes a singer's voice, mixes it with some background music, and renders it on a single set of speakers.

In the examples above, these *multi-content* devices support a fixed number of input streams. However, there are some devices that support an arbitrary number of input streams. For example, a PC can take some video content and mix it with the PC's display and render it in its own PC window. Depending on the processing power of the PC, it can mix together and render a variable number of video-windows.

In these examples, some advanced devices have the ability to control the rendering characteristics of each input item independently from each other as well as the post-mixed stream. For example, with a karaoke device, the volume of the singer's voice and the volume of the background music can be adjusted independently. Additionally, the volume of the post-mixed stream (that is: singer + background) can be adjusted as a whole without affecting the relative settings of each individual input stream.

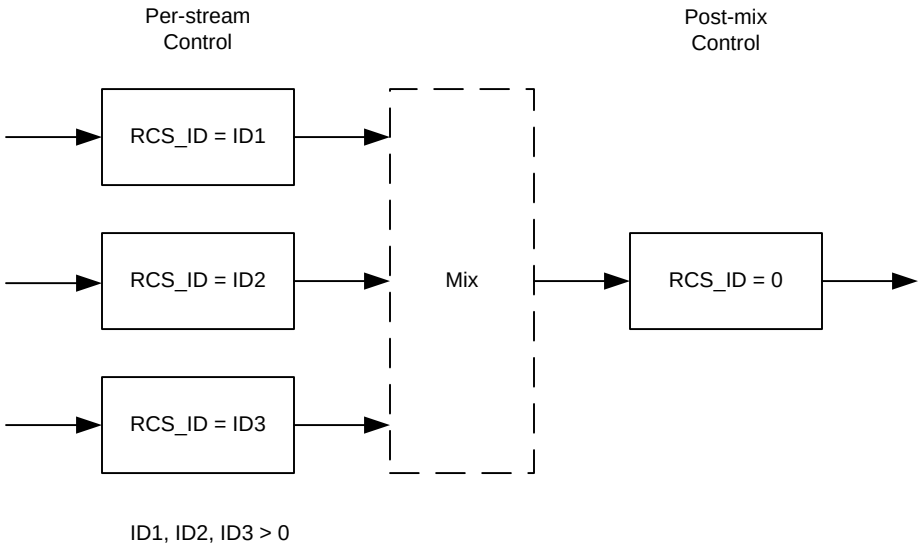

Figure 4: Virtual Instances of RCS

In order to support these types of devices, it is necessary for the RenderingControl service (RCS) to support multiple virtual instances of the service. As shown in Figure 4, a full-featured, high-end device that supports multiple input streams MUST assign a unique virtual instance of the RenderingControl service to each input stream. Each of these instances is assigned a unique ID (labeled RCS_ID in Figure 4) which can be used to adjust the rendering characteristics of its associated content without affecting any of the other input streams. Additionally, a default instance (ID=0) is assigned to control the rendering characteristics of the post-mixed stream (that is: all input content as a whole).

*InstanceID*s are used by control points when invoking RenderingControl service actions. The *InstanceID* argument (included with each action) allows the control point to indicate on which stream the invoked action MUST be applied. In order to control the rendering characteristics of an individual input stream (independently from all of the other streams) the control point uses the *InstanceID* associated with that stream. In order to control the rendering characteristics of the post-mixed stream, the control point uses *InstanceID* = 0.

New virtual instances of the RenderingControl service (that is: new *InstanceID*s) are allocated outside of the RenderingControl service using some external mechanism. As of this writing, only one allocation mechanism is defined. As described in the MediaRenderer device template, the device's *ConnectionManager::PrepareForConnection*() action assigns an *InstanceID* to the input stream (that is: each connection) that is being prepared. The number of instances that a device can support depends on the device's implementation. (Refer to the MediaRenderer device templates for additional information).

As defined by the UPnP Architecture, a device's description document contains a single service description document for each (physical) service that is implemented by the device. However, when a device supports

multiple virtual instances of the RenderingControl service, all of these virtual instances are represented by the service description document for the one (physical) RenderingControl service. In this case, the RenderingControl service description document reflects the actions and state variables supported by *InstanceID* = 0. All other non-zero virtual instances MUST support a subset of the actions and state variables supported by *InstanceID* = 0. However, each non-zero instance MAY support a different subset of *InstanceID* = 0 than the other non-zero virtual instances. For those state variables that are supported by a non-zero instance, each instance MUST support the identical *allowedValueList* and/or *allowedValueRange* as *InstanceID* = 0. If an unsupported action is invoked on a non-zero instance, the action will return error code 401 (Invalid Action).

As described in the MediaRenderer device template, a rendering device that contains multiple, independent rendering hardware (for example, two independent display screen, or two independent sets of output speakers) MUST be modeled as multiple instantiations of the MediaRenderer device, each with its own RenderingControl, ConnectionManager, and (OPTIONAL) AVTransport services. In other words, from an UPnP AV modeling point of view, each output on a physical rendering device is treated as a completely independent Media Rendering device. Refer to the MediaRenderer device template for more information.

2.5.2 Presets

Named presets allow a control point to put the device in a predetermined state in which certain state variables are set to predefined values. The set of currently available presets is listed in the *PresetNameList* state variable. Since a device is permitted to add or remove support for individual presets (for example, in conjunction with a vendor–defined action), the *PresetNameList* state variable is (indirectly) evented as described in Section 2.3, "Eventing and Moderation." Additionally, a control point can use the *ListPresets()* action to obtain an up to date list of supported presets.

A user can select one of the supported presets using the *SelectPreset()* action. This causes the device to set itself to a known state as defined by the selected preset. The exact definition of each preset is device-specific.

2.5.3 Controlling the Display of Visual Content

The RenderingControl service exposes a number of state variables that allow control points to control the appearance of visual content. These include such characteristics as brightness, contrast, color intensity, etc. In order to control these characteristics, the control point simply invokes the appropriate action. For most of these actions, the desired setting of the display characteristics is passed in by the control point. In most cases, this argument is a positive number between 0 and some device-specific maximum value. Each incremental value (that is: an increase or decrease by one) corresponds to the smallest amount of change supported by the device.

Determine the current Brightness setting of the main display:

- Invoke *GetBrightness()* with an *InstanceID* of zero.
- A return value of 13 indicates that the display's *Brightness* is currently set to 13 steps (that is: 13 device-specific increments) above the dimmest setting that is supported by the device.

Set the Brightness of the PIP display to the dimmest setting supported by the device:

- Invoke *SetBrightness()* with the PIP's *InstanceID* and a *Brightness* setting of 0.

2.5.4 Controlling Audio Content

The RenderingControl service exposes a set of state variables that can be used to control the audio output of a device. These include various characteristics such as volume, mute, and loudness. However, unlike most visual content, audio content is typically composed of one or more channels (for example, a left and right channel). The RenderingControl service allows control points to control each of these channels independently or as a whole. To accomplish this, it is necessary for the control point to identify the channel that is to be controlled. This is accomplished via the *Channel* argument included in each action that is associated with the audio portion of an input stream. Each channel is uniquely named as described in

Section 2.2.19, "*A_ARG_TYPE_Channel*." The *Master* channel allows a control point to control the audio content as a whole. The *Master* control influences all audio channels at the same time. Note however that it is conceptually a separate volume control (in each channel) and MUST NOT affect the current settings of the *Volume* state variables for the individual channels. The following figure provides an example for a 6-channel (LF, RF, CF, LFE, LS, and RS) volume control. The *Master* channel is actually a gang-coupled 6-channel volume control that impacts all channels at the same time with the same amount of volume change.

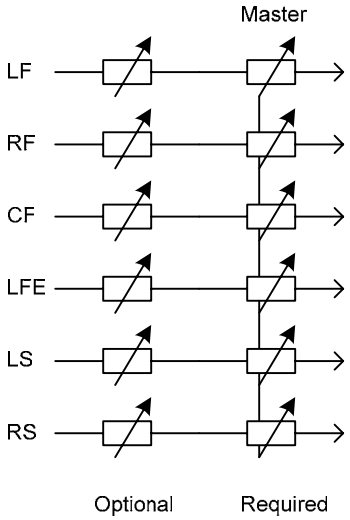

Figure 5: 6-channel Volume Control

When controlling the volume of a particular channel, control points can choose between two different representations of the volume setting. One representation uses the *Volume* state variable and the other representation uses the *VolumeDB* state variable. As described in Section 2.2.27.2, "*Volume* and *VolumeDB*," the *Volume* state variable represents volume as a contiguous set of *positions* numbered from 0 to some device-specific maximum, and the *VolumeDB* state variable represents volume in units of 1/256 of a decibel (dB). Two pairs of actions (one pair for each representation) are provided to get and set the volume of a channel.

The following example describes some scenarios that might help clarify the use of the *Get/SetVolume()* and *Get/SetVolumeDB()* actions.

Consider a volume control that is implemented as follows:

- Implemented Channels: "*Master*", "*LF*", "*RF*", "*CF*", "*LFE*", "*LS*", "*RS*"
- All Channels: 46 positions (N=45), end-of-scale mute implemented, *MinValue* = -72 dB, *MaxValue* = 0 dB, 1 dB resolution between 0 dB and -24 dB, 2 dB resolution between -24 dB and -48 dB, 3 dB resolution between -48 dB and -72 dB. (See Figure 3 above). The *Master* control is used to control the overall volume setting of the device whereas all other controls are intended to balance the different channels.
- At first power-up, the initial settings are: *Master* set to position 17 (-30 dB), all other channels set to position 32 (-12 dB)

Note: The current specification does not allow for different ranges for the controls on a per-channel basis. The Service Description contains only one entry for the *Volume* state variable. Therefore it is assumed that all volume controls are created equal for all channels and also for all virtual instances of the RenderingControl service. A future version of this specification will remedy this limitation.

Set the volume of the audio content (as a whole) to the quietest setting:

- Invoke the *SetVolume()* action with the *Channel* argument set to "*Master*" and the *DesiredVolume* argument set to 0.

Set the volume of the audio content (as a whole) 20 notches/steps higher than the current setting:

- Invoke the *GetVolume()* action with the *Channel* argument set to "*Master*". As a result of the previous example, the *CurrentVolume* out argument returns a value of 0 indicating that the audio content is being rendered at volume position 0; that is: the quietest setting supported by the device. A *GetVolumeDB()* action with the *Channel* argument set to "*Master*" will now return -18432 (0xB800 – increments of 1/256dB) in the *CurrentVolume* OUT argument, which corresponds to -72 dB. A *GetVolumeDB()* action with the *Channel* argument set to any other channel will still return -12 dB (0xF400 – increments of 1/256 dB)
- Invoke the *SetVolume()* action with the *Channel* argument set to "*Master*" and the *DesiredVolume* argument set to 20 (0 + 20). This corresponds to the 20th quietest setting supported by the device. A *GetVolumeDB()* action with the *Channel* argument set to "*Master*" will now return -6144 (0xE800) in the *CurrentVolume* OUT argument, which corresponds to -24 dB.

Set the volume of the Center channel 5dB higher than the *Master* channel:

- Invoke the *GetVolumeDB()* action with the *Channel* argument set to "*CF*" (for the Center Front channel). In this example, the *CurrentVolume* OUT argument still returns a value of -3072 (0xF400), which indicates that the audio content on the Center Front channel is being rendered at -12 dB, relative to the *Master* channel.
- Invoke the *SetVolumeDB()* action with the *Channel* argument set to "*CF*" and the *DesiredVolume* argument set to -1792 (0xF900) (-3072 + 1280 increments of 1/256dB) , which corresponds to -7 dB.

Double the volume of the entire audio content:

(Note: Increasing the volume by 6dB doubles the volume level.)

- Invoke the *GetVolumeDB()* action with the *Channel* argument set to "*Master*". Based on the previous example, the *CurrentVolume* OUT argument returned -6144 (0xE800), which indicates that the overall volume level is at -24dB.
- Invoke the *SetVolumeDB()* action with the *Channel* argument set to "*Master*" and the *DesiredVolume* argument set to -4608 (0xEE00) (-6144 + 1536 increments of 1/256dB), which corresponds to -18dB (-24+6).

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>ListPresets</name>
         <argumentList>
            <argument>
               <name>InstanceID</name>
               <direction>in</direction>
               <relatedStateVariable>
                  A_ARG_TYPE_InstanceID
               </relatedStateVariable>
            </argument>
            <argument>
               <name>CurrentPresetNameList</name>
               <direction>out</direction>
               <relatedStateVariable>
                  PresetNameList
               </relatedStateVariable>
            </argument>
         </argumentList>
      </action>
      <action>
         <name>SelectPreset</name>
         <argumentList>
            <argument>
               <name>InstanceID</name>
               <direction>in</direction>
               <relatedStateVariable>
                  A_ARG_TYPE_InstanceID
               </relatedStateVariable>
            </argument>
            <argument>
               <name>PresetName</name>
               <direction>in</direction>
               <relatedStateVariable>
                  A_ARG_TYPE_PresetName
               </relatedStateVariable>
            </argument>
         </argumentList>
      </action>
      <action>
         <name>GetBrightness</name>
         <argumentList>
            <argument>
               <name>InstanceID</name>
               <direction>in</direction>
               <relatedStateVariable>
                  A_ARG_TYPE_InstanceID
               </relatedStateVariable>
```

```
            </argument>
            <argument>
               <name>CurrentBrightness</name>
               <direction>out</direction>
               <relatedStateVariable>
                  Brightness
               </relatedStateVariable>
            </argument>
         </argumentList>
      </action>
      <action>
         <name>SetBrightness</name>
         <argumentList>
            <argument>
               <name>InstanceID</name>
               <direction>in</direction>
               <relatedStateVariable>
                  A_ARG_TYPE_InstanceID
               </relatedStateVariable>
            </argument>
            <argument>
               <name>DesiredBrightness</name>
               <direction>in</direction>
               <relatedStateVariable>
                  Brightness
               </relatedStateVariable>
            </argument>
         </argumentList>
      </action>
      <action>
         <name>GetContrast</name>
         <argumentList>
            <argument>
               <name>InstanceID</name>
               <direction>in</direction>
               <relatedStateVariable>
                  A_ARG_TYPE_InstanceID
               </relatedStateVariable>
            </argument>
            <argument>
               <name>CurrentContrast</name>
               <direction>out</direction>
               <relatedStateVariable>
                  Contrast
               </relatedStateVariable>
            </argument>
         </argumentList>
      </action>
      <action>
         <name>SetContrast</name>
         <argumentList>
            <argument>
               <name>InstanceID</name>
               <direction>in</direction>
               <relatedStateVariable>
                  A_ARG_TYPE_InstanceID
               </relatedStateVariable>
```

```
            </argument>
            <argument>
               <name>DesiredContrast</name>
               <direction>in</direction>
               <relatedStateVariable>
                  Contrast
               </relatedStateVariable>
            </argument>
         </argumentList>
      </action>
      <action>
         <name>GetSharpness</name>
         <argumentList>
            <argument>
               <name>InstanceID</name>
               <direction>in</direction>
               <relatedStateVariable>
                  A_ARG_TYPE_InstanceID
               </relatedStateVariable>
            </argument>
            <argument>
               <name>CurrentSharpness</name>
               <direction>out</direction>
               <relatedStateVariable>
                  Sharpness
               </relatedStateVariable>
            </argument>
         </argumentList>
      </action>
      <action>
         <name>SetSharpness</name>
         <argumentList>
            <argument>
               <name>InstanceID</name>
               <direction>in</direction>
               <relatedStateVariable>
                  A_ARG_TYPE_InstanceID
               </relatedStateVariable>
            </argument>
            <argument>
               <name>DesiredSharpness</name>
               <direction>in</direction>
               <relatedStateVariable>
                  Sharpness
               </relatedStateVariable>
            </argument>
         </argumentList>
      </action>
      <action>
         <name>GetRedVideoGain</name>
         <argumentList>
            <argument>
               <name>InstanceID</name>
               <direction>in</direction>
               <relatedStateVariable>
                  A_ARG_TYPE_InstanceID
               </relatedStateVariable>
```

```
            </argument>
            <argument>
               <name>CurrentRedVideoGain</name>
               <direction>out</direction>
               <relatedStateVariable>
                  RedVideoGain
               </relatedStateVariable>
            </argument>
         </argumentList>
      </action>
      <action>
         <name>SetRedVideoGain</name>
         <argumentList>
            <argument>
               <name>InstanceID</name>
               <direction>in</direction>
               <relatedStateVariable>
                  A_ARG_TYPE_InstanceID
               </relatedStateVariable>
            </argument>
            <argument>
               <name>DesiredRedVideoGain</name>
               <direction>in</direction>
               <relatedStateVariable>
                  RedVideoGain
               </relatedStateVariable>
            </argument>
         </argumentList>
      </action>
      <action>
         <name>GetGreenVideoGain</name>
         <argumentList>
            <argument>
               <name>InstanceID</name>
               <direction>in</direction>
               <relatedStateVariable>
                  A_ARG_TYPE_InstanceID
               </relatedStateVariable>
            </argument>
            <argument>
               <name>CurrentGreenVideoGain</name>
               <direction>out</direction>
               <relatedStateVariable>
                  GreenVideoGain
               </relatedStateVariable>
            </argument>
         </argumentList>
      </action>
      <action>
         <name>SetGreenVideoGain</name>
         <argumentList>
            <argument>
               <name>InstanceID</name>
               <direction>in</direction>
               <relatedStateVariable>
                  A_ARG_TYPE_InstanceID
               </relatedStateVariable>
```

```
            </argument>
            <argument>
               <name>DesiredGreenVideoGain</name>
               <direction>in</direction>
               <relatedStateVariable>
                  GreenVideoGain
               </relatedStateVariable>
            </argument>
         </argumentList>
      </action>
      <action>
         <name>GetBlueVideoGain</name>
         <argumentList>
            <argument>
               <name>InstanceID</name>
               <direction>in</direction>
               <relatedStateVariable>
                  A_ARG_TYPE_InstanceID
               </relatedStateVariable>
            </argument>
            <argument>
               <name>CurrentBlueVideoGain</name>
               <direction>out</direction>
               <relatedStateVariable>
                  BlueVideoGain
               </relatedStateVariable>
            </argument>
         </argumentList>
      </action>
      <action>
         <name>SetBlueVideoGain</name>
         <argumentList>
            <argument>
               <name>InstanceID</name>
               <direction>in</direction>
               <relatedStateVariable>
                  A_ARG_TYPE_InstanceID
               </relatedStateVariable>
            </argument>
            <argument>
               <name>DesiredBlueVideoGain</name>
               <direction>in</direction>
               <relatedStateVariable>
                  BlueVideoGain
               </relatedStateVariable>
            </argument>
         </argumentList>
      </action>
      <action>
         <name>GetRedVideoBlackLevel</name>
         <argumentList>
            <argument>
               <name>InstanceID</name>
               <direction>in</direction>
               <relatedStateVariable>
                  A_ARG_TYPE_InstanceID
               </relatedStateVariable>
```

```
            </argument>
            <argument>
               <name>CurrentRedVideoBlackLevel</name>
               <direction>out</direction>
               <relatedStateVariable>
                  RedVideoBlackLevel
               </relatedStateVariable>
            </argument>
         </argumentList>
      </action>
      <action>
         <name>SetRedVideoBlackLevel</name>
         <argumentList>
            <argument>
               <name>InstanceID</name>
               <direction>in</direction>
               <relatedStateVariable>
                  A_ARG_TYPE_InstanceID
               </relatedStateVariable>
            </argument>
            <argument>
               <name>DesiredRedVideoBlackLevel</name>
               <direction>in</direction>
               <relatedStateVariable>
                  RedVideoBlackLevel
               </relatedStateVariable>
            </argument>
         </argumentList>
      </action>
      <action>
         <name>GetGreenVideoBlackLevel</name>
         <argumentList>
            <argument>
               <name>InstanceID</name>
               <direction>in</direction>
               <relatedStateVariable>
                  A_ARG_TYPE_InstanceID
               </relatedStateVariable>
            </argument>
            <argument>
               <name>CurrentGreenVideoBlackLevel</name>
               <direction>out</direction>
               <relatedStateVariable>
                  GreenVideoBlackLevel
               </relatedStateVariable>
            </argument>
         </argumentList>
      </action>
      <action>
         <name>SetGreenVideoBlackLevel</name>
         <argumentList>
            <argument>
               <name>InstanceID</name>
               <direction>in</direction>
               <relatedStateVariable>
                  A_ARG_TYPE_InstanceID
               </relatedStateVariable>
```

```
            </argument>
            <argument>
               <name>DesiredGreenVideoBlackLevel</name>
               <direction>in</direction>
               <relatedStateVariable>
                  GreenVideoBlackLevel
               </relatedStateVariable>
            </argument>
         </argumentList>
      </action>
      <action>
         <name>GetBlueVideoBlackLevel</name>
         <argumentList>
            <argument>
               <name>InstanceID</name>
               <direction>in</direction>
               <relatedStateVariable>
                  A_ARG_TYPE_InstanceID
               </relatedStateVariable>
            </argument>
            <argument>
               <name>CurrentBlueVideoBlackLevel</name>
               <direction>out</direction>
               <relatedStateVariable>
                  BlueVideoBlackLevel
               </relatedStateVariable>
            </argument>
         </argumentList>
      </action>
      <action>
         <name>SetBlueVideoBlackLevel</name>
         <argumentList>
            <argument>
               <name>InstanceID</name>
               <direction>in</direction>
               <relatedStateVariable>
                  A_ARG_TYPE_InstanceID
               </relatedStateVariable>
            </argument>
            <argument>
               <name>DesiredBlueVideoBlackLevel</name>
               <direction>in</direction>
               <relatedStateVariable>
                  BlueVideoBlackLevel
               </relatedStateVariable>
            </argument>
         </argumentList>
      </action>
      <action>
         <name>GetColorTemperature</name>
         <argumentList>
            <argument>
               <name>InstanceID</name>
               <direction>in</direction>
               <relatedStateVariable>
                  A_ARG_TYPE_InstanceID
               </relatedStateVariable>
```

```
            </argument>
            <argument>
               <name>CurrentColorTemperature</name>
               <direction>out</direction>
               <relatedStateVariable>
                  ColorTemperature
               </relatedStateVariable>
            </argument>
         </argumentList>
      </action>
      <action>
         <name>SetColorTemperature</name>
         <argumentList>
            <argument>
               <name>InstanceID</name>
               <direction>in</direction>
               <relatedStateVariable>
                  A_ARG_TYPE_InstanceID
               </relatedStateVariable>
            </argument>
            <argument>
               <name>DesiredColorTemperature</name>
               <direction>in</direction>
               <relatedStateVariable>
                  ColorTemperature
               </relatedStateVariable>
            </argument>
         </argumentList>
      </action>
      <action>
         <name>GetHorizontalKeystone</name>
         <argumentList>
            <argument>
               <name>InstanceID</name>
               <direction>in</direction>
               <relatedStateVariable>
                  A_ARG_TYPE_InstanceID
               </relatedStateVariable>
            </argument>
            <argument>
               <name>CurrentHorizontalKeystone</name>
               <direction>out</direction>
               <relatedStateVariable>
                  HorizontalKeystone
               </relatedStateVariable>
            </argument>
         </argumentList>
      </action>
      <action>
         <name>SetHorizontalKeystone</name>
         <argumentList>
            <argument>
               <name>InstanceID</name>
               <direction>in</direction>
               <relatedStateVariable>
                  A_ARG_TYPE_InstanceID
               </relatedStateVariable>
```

```
            </argument>
            <argument>
               <name>DesiredHorizontalKeystone</name>
               <direction>in</direction>
               <relatedStateVariable>
                  HorizontalKeystone
               </relatedStateVariable>
            </argument>
         </argumentList>
      </action>
      <action>
         <name>GetVerticalKeystone</name>
         <argumentList>
            <argument>
               <name>InstanceID</name>
               <direction>in</direction>
               <relatedStateVariable>
                  A_ARG_TYPE_InstanceID
               </relatedStateVariable>
            </argument>
            <argument>
               <name>CurrentVerticalKeystone</name>
               <direction>out</direction>
               <relatedStateVariable>
                  VerticalKeystone
               </relatedStateVariable>
            </argument>
         </argumentList>
      </action>
      <action>
         <name>SetVerticalKeystone</name>
         <argumentList>
            <argument>
               <name>InstanceID</name>
               <direction>in</direction>
               <relatedStateVariable>
                  A_ARG_TYPE_InstanceID
               </relatedStateVariable>
            </argument>
            <argument>
               <name>DesiredVerticalKeystone</name>
               <direction>in</direction>
               <relatedStateVariable>
                  VerticalKeystone
               </relatedStateVariable>
            </argument>
         </argumentList>
      </action>
      <action>
         <name>GetMute</name>
         <argumentList>
            <argument>
               <name>InstanceID</name>
               <direction>in</direction>
               <relatedStateVariable>
                  A_ARG_TYPE_InstanceID
               </relatedStateVariable>
```

```
            </argument>
            <argument>
               <name>Channel</name>
               <direction>in</direction>
               <relatedStateVariable>
                  A_ARG_TYPE_Channel
               </relatedStateVariable>
            </argument>
            <argument>
               <name>CurrentMute</name>
               <direction>out</direction>
               <relatedStateVariable>
                  Mute
               </relatedStateVariable>
            </argument>
         </argumentList>
      </action>
      <action>
         <name>SetMute</name>
         <argumentList>
            <argument>
               <name>InstanceID</name>
               <direction>in</direction>
               <relatedStateVariable>
                  A_ARG_TYPE_InstanceID
               </relatedStateVariable>
            </argument>
            <argument>
               <name>Channel</name>
               <direction>in</direction>
               <relatedStateVariable>
                  A_ARG_TYPE_Channel
               </relatedStateVariable>
            </argument>
            <argument>
               <name>DesiredMute</name>
               <direction>in</direction>
               <relatedStateVariable>
                  Mute
               </relatedStateVariable>
            </argument>
         </argumentList>
      </action>
      <action>
         <name>GetVolume</name>
         <argumentList>
            <argument>
               <name>InstanceID</name>
               <direction>in</direction>
               <relatedStateVariable>
                  A_ARG_TYPE_InstanceID
               </relatedStateVariable>
            </argument>
            <argument>
               <name>Channel</name>
               <direction>in</direction>
               <relatedStateVariable>
```

```
                A_ARG_TYPE_Channel
            </relatedStateVariable>
        </argument>
        <argument>
            <name>CurrentVolume</name>
            <direction>out</direction>
            <relatedStateVariable>
                Volume
            </relatedStateVariable>
        </argument>
    </argumentList>
</action>
<action>
    <name>SetVolume</name>
    <argumentList>
        <argument>
            <name>InstanceID</name>
            <direction>in</direction>
            <relatedStateVariable>
                A_ARG_TYPE_InstanceID
            </relatedStateVariable>
        </argument>
        <argument>
            <name>Channel</name>
            <direction>in</direction>
            <relatedStateVariable>
                A_ARG_TYPE_Channel
            </relatedStateVariable>
        </argument>
        <argument>
            <name>DesiredVolume</name>
            <direction>in</direction>
            <relatedStateVariable>
                Volume
            </relatedStateVariable>
        </argument>
    </argumentList>
</action>
<action>
    <name>GetVolumeDB</name>
    <argumentList>
        <argument>
            <name>InstanceID</name>
            <direction>in</direction>
            <relatedStateVariable>
                A_ARG_TYPE_InstanceID
            </relatedStateVariable>
        </argument>
        <argument>
            <name>Channel</name>
            <direction>in</direction>
            <relatedStateVariable>
                A_ARG_TYPE_Channel
            </relatedStateVariable>
        </argument>
        <argument>
            <name>CurrentVolume</name>
```

```
            <direction>out</direction>
            <relatedStateVariable>
               VolumeDB
            </relatedStateVariable>
         </argument>
      </argumentList>
   </action>
   <action>
      <name>SetVolumeDB</name>
      <argumentList>
         <argument>
            <name>InstanceID</name>
            <direction>in</direction>
            <relatedStateVariable>
               A_ARG_TYPE_InstanceID
            </relatedStateVariable>
         </argument>
         <argument>
            <name>Channel</name>
            <direction>in</direction>
            <relatedStateVariable>
               A_ARG_TYPE_Channel
            </relatedStateVariable>
         </argument>
         <argument>
            <name>DesiredVolume</name>
            <direction>in</direction>
            <relatedStateVariable>
               VolumeDB
            </relatedStateVariable>
         </argument>
      </argumentList>
   </action>
   <action>
      <name>GetVolumeDBRange</name>
      <argumentList>
         <argument>
            <name>InstanceID</name>
            <direction>in</direction>
            <relatedStateVariable>
               A_ARG_TYPE_InstanceID
            </relatedStateVariable>
         </argument>
         <argument>
            <name>Channel</name>
            <direction>in</direction>
            <relatedStateVariable>
               A_ARG_TYPE_Channel
            </relatedStateVariable>
         </argument>
         <argument>
            <name>MinValue</name>
            <direction>out</direction>
            <relatedStateVariable>
               VolumeDB
            </relatedStateVariable>
         </argument>
```

```
            <argument>
               <name>MaxValue</name>
               <direction>out</direction>
               <relatedStateVariable>
                  VolumeDB
               </relatedStateVariable>
            </argument>
         </argumentList>
      </action>
      <action>
         <name>GetLoudness</name>
         <argumentList>
            <argument>
               <name>InstanceID</name>
               <direction>in</direction>
               <relatedStateVariable>
                  A_ARG_TYPE_InstanceID
               </relatedStateVariable>
            </argument>
            <argument>
               <name>Channel</name>
               <direction>in</direction>
               <relatedStateVariable>
                  A_ARG_TYPE_Channel
               </relatedStateVariable>
            </argument>
            <argument>
               <name>CurrentLoudness</name>
               <direction>out</direction>
               <relatedStateVariable>
                  Loudness
               </relatedStateVariable>
            </argument>
         </argumentList>
      </action>
      <action>
         <name>SetLoudness</name>
         <argumentList>
            <argument>
               <name>InstanceID</name>
               <direction>in</direction>
               <relatedStateVariable>
                  A_ARG_TYPE_InstanceID
               </relatedStateVariable>
            </argument>
            <argument>
               <name>Channel</name>
               <direction>in</direction>
               <relatedStateVariable>
                  A_ARG_TYPE_Channel
               </relatedStateVariable>
            </argument>
            <argument>
               <name>DesiredLoudness</name>
               <direction>in</direction>
               <relatedStateVariable>
                  Loudness
```

```
            </relatedStateVariable>
         </argument>
      </argumentList>
   </action>
   <action>
      <name>GetStateVariables</name>
      <argumentList>
         <argument>
            <name>InstanceID</name>
            <direction>in</direction>
            <relatedStateVariable>
               A_ARG_TYPE_InstanceID
            </relatedStateVariable>
         </argument>
         <argument>
            <name>StateVariableList</name>
            <direction>in</direction>
            <relatedStateVariable>
               A_ARG_TYPE_StateVariableList
            </relatedStateVariable>
         </argument>
         <argument>
            <name>StateVariableValuePairs</name>
            <direction>in</direction>
            <relatedStateVariable>
               A_ARG_TYPE_StateVariableValuePairs
            </relatedStateVariable>
         </argument>
      </argumentList>
   </action>
   <action>
      <name>SetStateVariables</name>
      <argumentList>
         <argument>
            <name>InstanceID</name>
            <direction>in</direction>
            <relatedStateVariable>
               A_ARG_TYPE_InstanceID
            </relatedStateVariable>
         </argument>
         <argument>
            <name>RenderingControlUDN</name>
            <direction>in</direction>
            <relatedStateVariable>
               A_ARG_TYPE_DeviceUDN
            </relatedStateVariable>
         </argument>
         <argument>
            <name>ServiceType</name>
            <direction>in</direction>
            <relatedStateVariable>
               A_ARG_TYPE_ServiceType
            </relatedStateVariable>
         </argument>
         <argument>
            <name>ServiceId</name>
            <direction>in</direction>
```

```
            <relatedStateVariable>
               A_ARG_TYPE_ServiceID
            </relatedStateVariable>
         </argument>
         <argument>
            <name>StateVariableValuePairs</name>
            <direction>in</direction>
            <relatedStateVariable>
               A_ARG_TYPE_StateVariableValuePairs
            </relatedStateVariable>
         </argument>
         <argument>
            <name>StateVariableList</name>
            <direction>out</direction>
            <relatedStateVariable>
               A_ARG_TYPE_StateVariableList
            </relatedStateVariable>
         </argument>
      </argumentList>
   </action>
   Declarations for other actions added by UPnP vendor
   (if any) go here
</actionList>
<serviceStateTable>
   <stateVariable sendEvents="no">
      <name>PresetNameList</name>
      <dataType>string</dataType>
   </stateVariable>
   <stateVariable sendEvents="yes">
      <name>LastChange</name>
      <dataType>string</dataType>
   </stateVariable>
   <stateVariable sendEvents="no">
      <name>Brightness</name>
      <dataType>ui2</dataType>
      <allowedValueRange>
         <minimum>0</minimum>
         <maximum>Vendor defined</maximum>
         <step>1</step>
      </allowedValueRange>
   </stateVariable>
   <stateVariable sendEvents="no">
      <name>Contrast</name>
      <dataType>ui2</dataType>
      <allowedValueRange>
         <minimum>0</minimum>
         <maximum>Vendor defined</maximum>
         <step>1</step>
      </allowedValueRange>
   </stateVariable>
   <stateVariable sendEvents="no">
      <name>Sharpness</name>
      <dataType>ui2</dataType>
      <allowedValueRange>
         <minimum>0</minimum>
         <maximum>Vendor defined</maximum>
         <step>1</step>
```

```
      </allowedValueRange>
   </stateVariable>
   <stateVariable sendEvents="no">
      <name>RedVideoGain</name>
      <dataType>ui2</dataType>
      <allowedValueRange>
         <minimum>0</minimum>
         <maximum>Vendor defined</maximum>
         <step>1</step>
      </allowedValueRange>
   </stateVariable>
   <stateVariable sendEvents="no">
      <name>GreenVideoGain</name>
      <dataType>ui2</dataType>
      <allowedValueRange>
         <minimum>0</minimum>
         <maximum>Vendor defined</maximum>
         <step>1</step>
      </allowedValueRange>
   </stateVariable>
   <stateVariable sendEvents="no">
      <name>BlueVideoGain</name>
      <dataType>ui2</dataType>
      <allowedValueRange>
         <minimum>0</minimum>
         <maximum>Vendor defined</maximum>
         <step>1</step>
      </allowedValueRange>
   </stateVariable>
   <stateVariable sendEvents="no">
      <name>RedVideoBlackLevel</name>
      <dataType>ui2</dataType>
      <allowedValueRange>
         <minimum>0</minimum>
         <maximum>Vendor defined</maximum>
         <step>1</step>
      </allowedValueRange>
   </stateVariable>
   <stateVariable sendEvents="no">
      <name>GreenVideoBlackLevel</name>
      <dataType>ui2</dataType>
      <allowedValueRange>
         <minimum>0</minimum>
         <maximum>Vendor defined</maximum>
         <step>1</step>
      </allowedValueRange>
   </stateVariable>
   <stateVariable sendEvents="no">
      <name>BlueVideoBlackLevel</name>
      <dataType>ui2</dataType>
      <allowedValueRange>
         <minimum>0</minimum>
         <maximum>Vendor defined</maximum>
         <step>1</step>
      </allowedValueRange>
   </stateVariable>
   <stateVariable sendEvents="no">
```

```
      <name>ColorTemperature</name>
      <dataType>ui2</dataType>
      <allowedValueRange>
         <minimum>0</minimum>
         <maximum>Vendor defined</maximum>
         <step>1</step>
      </allowedValueRange>
   </stateVariable>
   <stateVariable sendEvents="no">
      <name>HorizontalKeystone</name>
      <dataType>i2</dataType>
      <allowedValueRange>
         <minimum>Vendor defined (MUST be <= 0)</minimum>
         <maximum>Vendor defined</maximum>
         <step>1</step>
      </allowedValueRange>
   </stateVariable>
   <stateVariable sendEvents="no">
      <name>VerticalKeystone</name>
      <dataType>i2</dataType>
      <allowedValueRange>
         <minimum>Vendor defined (MUST be <= 0)</minimum>
         <maximum>Vendor defined</maximum>
         <step>1</step>
      </allowedValueRange>
   </stateVariable>
   <stateVariable sendEvents="no">
      <name>Mute</name>
      <dataType>boolean</dataType>
   </stateVariable>
   <stateVariable sendEvents="no">
      <name>Volume</name>
      <dataType>ui2</dataType>
      <allowedValueRange>
         <minimum>0</minimum>
         <maximum>Vendor defined</maximum>
         <step>1</step>
      </allowedValueRange>
   </stateVariable>
   <stateVariable sendEvents="no">
      <name>VolumeDB</name>
      <dataType>i2</dataType>
      <allowedValueRange>
         <minimum>Vendor defined</minimum>
         <maximum>Vendor defined</maximum>
         <step>Vendor defined</step>
      </allowedValueRange>
   </stateVariable>
   <stateVariable sendEvents="no">
      <name>Loudness</name>
      <dataType>boolean</dataType>
   </stateVariable>
   <stateVariable sendEvents="no">
      <name>A_ARG_TYPE_Channel</name>
      <dataType>string</dataType>
      <allowedValueList>
         <allowedValue>Master</allowedValue>
```

```
            <allowedValue>LF</allowedValue>
            <allowedValue>RF</allowedValue>
            <allowedValue>CF</allowedValue>
            <allowedValue>LFE</allowedValue>
            <allowedValue>LS</allowedValue>
            <allowedValue>RS</allowedValue>
            <allowedValue>LFC</allowedValue>
            <allowedValue>RFC</allowedValue>
            <allowedValue>SD</allowedValue>
            <allowedValue>SL</allowedValue>
            <allowedValue>SR </allowedValue>
            <allowedValue>T</allowedValue>
            <allowedValue>B</allowedValue>
            <allowedValue>Vendor defined</allowedValue>
         </allowedValueList>
      </stateVariable>
      <stateVariable sendEvents="no">
         <name>A_ARG_TYPE_InstanceID</name>
         <dataType>ui4</dataType>
      </stateVariable>
      <stateVariable sendEvents="no">
         <name>A_ARG_TYPE_PresetName</name>
         <dataType>string</dataType>
         <allowedValueList>
            <allowedValue>FactoryDefaults</allowedValue>
            <allowedValue>InstallationDefaults</allowedValue>
            <allowedValue>Vendor defined</allowedValue>
         </allowedValueList>
      </stateVariable>
      <stateVariable sendEvents="no">
         <name>A_ARG_TYPE_DeviceUDN</name>
         <dataType>string</dataType>
      </stateVariable>
      <stateVariable sendEvents="no">
         <name>A_ARG_TYPE_ServiceType</name>
         <dataType>string</dataType>
      </stateVariable>
      <stateVariable sendEvents="no">
         <name>A_ARG_TYPE_ServiceID</name>
         <dataType>string</dataType>
      </stateVariable>
      <stateVariable sendEvents="no">
         <name>A_ARG_TYPE_StateVariableValuePairs</name>
         <dataType>string</dataType>
      </stateVariable>
      <stateVariable sendEvents="no">
         <name>A_ARG_TYPE_StateVariableList</name>
         <dataType>string</dataType>
      </stateVariable>
      Declarations for other state variables added by UPnP vendor
      (if any) go here
   </serviceStateTable>
</scpd>
```

4 Test

There are no semantic tests mandated for this service.