RenderingControl:3 Service

For UPnP Version 1.0
Status: Standardized DCP (SDCP)
Date: December 31, 2010
Service Template Version 1.01

Authors	Company
Alan Presser	Allegrosoft
Gary Langille	Echostar
Gerrie Shults	HP
Raj Bopardikar	Intel
John Ritchie	Intel
Mark Walker	Intel
Sungjoon Ahn	LG Electronics
Changhyun Kim	LG Electronics
Jack Unverferth	Microsoft
Keith Miller (Chair)	Nokia
Masatomo Hori	Panasonic
Matthew Ma	Panasonic
Wouter van der Beek	Philips
Wim Bronnenberg	Philips
Jeffrey Kang	Philips

Authors	Company
Geert Knapen	Philips
Russell Berkoff	Pioneer
Irene Shen	Pioneer
Russell Berkoff (Vice-Chair)	Samsung Electronics
Richard Bardini	Sony
Norifumi Kikkawa	Sony
Jonathan Tourzan	Sony
Yasuhiro Morioka	Toshiba

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

Contents

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
- Perform a set of content transforms, which, in addition to the above, also enables functionality for selecting content depended options, for example:
  - Selecting a specific sub-stream in a composite stream for rendering.
  - Turning subtitling on or off.
- 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 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 keywords “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 prohibited by 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 specification are printed using the **arch** character style [DEVICE].
- 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 **boolean** 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 Boolean 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 ({a,b,c}, {*x*, *y*, *z*}), where a, b, c, *x*, *y* and *z* are the types of the individual values in the subsequence and the subsequences 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 US-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* supersede XML rules for usage in documents, it only augments and complements them in important contexts that are out-of-scope for the XML specifications. For example, action arguments which refer to CDS properties, such as the *SearchCriteria* argument of the *Search()* action or the *Filter* argument of the *Browse()* action, MUST use the predefined namespace prefixes when referring to CDS properties (“upnp:”, “dc:”, etc).

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. For example, since the Scheduled Recording Service depends on and refers to the Content Directory Service, the predefined “srs:” namespace prefix is included. 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 element(s) (if any), its schema file name, versioning information (to be discussed in more detail below), and a link to the entry in Section 1.7, “References” for its associated schema.

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

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	
atrs	urn:schemas-upnp-org:av:AllowedTransformSettings	*AllowedTransformSettings* and *AllowedDefaultTransformSettings* state variables for RenderingControl	[RCS]
av	urn:schemas-upnp-org:av:av	Common data types for use in AV schemas	[AV-XSD]
avdt	urn:schemas-upnp-org:av:avdt	Datastructure Template	[AVDT]
avs	urn:schemas-upnp-org:av:avs	Common structures for use in AV schemas	[AVS-XSD]
avt-event	urn:schemas-upnp-org:metadata-1-0/AVT/	Evented *LastChange* state variable for AVTransport	[AVT]
cds-event	urn:schemas-upnp-org:av:cds-event	Evented *LastChange* state variable for ContentDirectory	[CDS]
cm-dciu	urn:schemas-upnp-org:av:cm-deviceClockInfoUpdates	Evented *DeviceClockInfoUpdates* state variable for ConnectionManager	[CM]
cm-ftrlst	urn:schemas-upnp-org:av:cm-featureList	*FeatureList* state variable for ConnectionManager	[CM]
didl-lite	urn:schemas-upnp-org:metadata-1-0/DIDL-Lite/	Structure and metadata for ContentDirectory	[CDS]
dmo	urn:schemas-upnp.org:av:dmo	Evented *DeviceMode* state variable for ContentDirectory	[CDS]
dmor	urn:schemas-upnp.org:av:dmor	*A_ARG_TYPE_DeviceModeRequest* state variable for ContentDirectory	[CDS]
dmos	urn:schemas-upnp.org:av:dmos	*DeviceModeStatus* state variable for ContentDirectory	[CDS]
pi	urn:schemas-upnp.org:av:pi	*PermissionsInfo* state variable for ContentDirectory	[CDS]
rcs-event	urn:schemas-upnp-org:metadata-1-0/RCS/	Evented *LastChange* state variable for RenderingControl	[RCS]
rii	urn:schemas-upnp-org:av:rii	*A_ARG_TYPE_RenderingInfoList* state variable for ConnectionManager	[CM]
rpl	urn:schemas-upnp-org:av:rpl	*A_ARG_TYPE_PlaylistInfo* state variable for AVTransport	[AVT]

Standard Name-space Prefix	Namespace Name	Namespace Description	Normative Definition Document Reference
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]
trs	urn:schemas-upnp-org:av:TransformSettings	*TransformSettings* and *DefaultTransformSettings* state variables for RenderingControl	[RCS]
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[1] ● Form 1, Form 2, Form3	Valid Root Element(s)	Schema Reference
		AV Working Committee Defined Namespaces	
atrs	• AllowedTransformSettings-v*n*-*yyyymmdd*.xsd • AllowedTransformSettings-v*n*.xsd • AllowedTransformSettings.xsd	`<TransformList>`	[ATRS-XSD]
av	• av-v*n*-*yyyymmdd*.xsd • av-v*n*.xsd • av.xsd	*n/a*	[AV-XSD]
avdt	• avdt-v*n*-*yyyymmdd*.xsd • avdt-v*n*.xsd • avdt.xsd	`<AVDT>`	[AVDT]

avs	• avs-v*n*-*yyyymmdd*.xsd
	• avs-v*n*.xsd
	• avs.xsd
		`<Capabilities>`
		`<Features>`
		`<stateVariableValuePairs>`
			[AVS-XSD]

avt-event	• avt-event-v*n*-*yyyymmdd*.xsd • avt-event-v*n*.xsd • avt-event.xsd	`<Event>`	[AVT-EVENT-XSD]
cds-event	• cds-event-v*n*-*yyyymmdd*.xsd • cds-event-vn.xsd • cds-event.xsd	`<StateEvent>`	[CDS-EVENT-XSD]

Standard Name-space Prefix	Relative URI and File Name[1] • Form 1, Form 2, Form3	Valid Root Element(s)	Schema Reference
cm-dciu	• cm-deviceClockInfoUpdates-*vn-yyyymmdd*.xsd • cm-deviceClockInfoUpdates -*vn*.xsd • cm-deviceClockInfoUpdates.xsd	`<DeviceClockInfoUpdates>`	[CM-DCIU-XSD]
cm-ftrlst	• cm-featureList-*vn-yyyymmdd*.xsd • cm-featureList-*vn*.xsd • cm-featureList.xsd	`<Features>`	[CM-FTRLST-XSD]
didl-lite	• didl-lite-vn-*yyyymmdd*.xsd • didl-lite-v*n*.xsd • didl-lite.xsd	`<DIDL-Lite>`	[DIDL-LITE-XSD]
dmo	• dmo-vn-*yyyymmdd*.xsd • dmo-v*n*.xsd • dmo.xsd	`<DeviceMode>`	[DMO-XSD]
dmor	• dmor-vn-*yyyymmdd*.xsd • dmor-v*n*.xsd • dmor.xsd	`<DeviceModeRequest>`	[DMOR-XSD]
dmos	• dmos-vn-*yyyymmdd*.xsd • dmos-v*n*.xsd • dmos.xsd	`<DeviceModeStatus>`	[DMOS-XSD]
pi	• pi-vn-*yyyymmdd*.xsd • pi-v*n*.xsd • pi.xsd	`<PermissionsInfo>`	[PI-XSD]
rcs-event	• rcs-event-vn-*yyyymmdd*.xsd • rcs-event-v*n*.xsd • rcs-event.xsd	`<Event>`	[RCS-EVENT-XSD]
rii	• rii-*vn-yyyymmdd*.xsd • rii-*vn*.xsd • rii.xsd	`<rendererInfo>`	[RII-XSD]
rpl	• rpl-*vn-yyyymmdd*.xsd • rpl-*vn*.xsd • rpl.xsd	`<PlaylistInfo>`	[RPL-XSD]
srs	• srs-vn-*yyyymmdd*.xsd • srs-v*n*.xsd • srs.xsd	`<srs>`	[SRS-XSD]
srs-event	• srs-event-vn-*yyyymmdd*.xsd • srs-event-v*n*.xsd • srs-event.xsd	`<StateEvent>`	[SRS-EVENT-XSD]

Standard Name-space Prefix	Relative URI and File Name[1] • Form 1, Form 2, Form3	Valid Root Element(s)	Schema Reference
trs	• TransformSettings-*vn*-*yyyymmdd*.xsd • TransformSettings-*vn*.xsd • TransformSettings.xsd	`<TransformSettings>`	[TRS-XSD]
upnp	• upnp-v*n*-*yyyymmdd*.xsd • upnp-v*n*.xsd • upnp.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]Absolute URIs are generated by prefixing the relative URIs with "http://www.upnp.org/schemas/av/".

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. However, 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	avt-event
ConnectionManager	*n/a*
ContentDirectory	didl-lite
MediaRenderer	*n/a*
MediaServer	*n/a*
RenderingControl	rcs-event

AV Specification Name	Default Namespace Prefix
ScheduledRecording	srs

1.5.2 Namespace Names, Namespace Versioning and Schema Versioning

The UPnP AV service specifications define several data structures (such as state variables and action arguments) whose format is an XML instance document that must comply with one or more specific XML namespaces. Each namespace is uniquely identified by an assigned namespace name. The namespaces that are defined by the AV Working Committee MUST be named by a URN. See Table 1-3, "Namespace Definitions" for a current list of namespace names. Additionally, each namespace corresponds to an XML schema document that provides a machine-readable representation of the associated namespace to enable automated validation of the XML (state variable or action parameter) instance documents.

Within an XML schema and XML instance document, the name of each corresponding namespace appears as the value of an `xmlns` attribute within the root element. Each `xmlns` attribute also includes a namespace prefix that is associated with that namespace in order to disambiguate (a.k.a. qualify) element and attribute names that are defined within different namespaces. The schemas that correspond to the listed namespaces are identified by URI values that are listed in the `schemaLocation` attribute also within the root element. (See Section 1.5.3, "Namespace Usage Examples")

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

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

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

where

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

Table 1-4, "Schema-related Information" identifies the URI formats for each of the namespaces that are currently defined by the UPnP AV Working Committee.

As an example, the original schema URI for the "rcs-event" namespace (that was released with the original publication of the UPnP AV service specifications in the year 2002) was "http://www.upnp.org/schemas/av/rcs-event-v1-20020625.xsd". When the UPnP AV service specifications were subsequently updated in the year 2006, the URI for the updated version of the "rcs-event" namespace was "http://www.upnp.org/schemas/av/rcs-event-v2-20060531.xsd". However, in 2006, the schema URI for the newly created "srs-event" namespace was "http://www.upnp.org/schemas/av/srs-event-v1-20060531.xsd". Note the version field for the "srs-event" schema is "v1" since it was first version of that namespace whereas the version field for the "rcs-event" schema is "v2" since it was the second version of that namespace.

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

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

where ***ver*** is described above.

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

Form 2 of the undated schema URI is always linked to the most recent release of the schema that represents the version of the namespace indicated by ***ver***. For example, the undated URI "…/av/rcs-event-v2.xsd" is linked to the most recent schema release of version 2 of the "rcs-event" namespace. Therefore, on May 31, 2006 (20060531), the undated schema URI was linked to the schema that is otherwise known as "…/av/rcs-event-v2-20060531.xsd". Furthermore, if the schema for version 2 of the "rcs-event" namespace was ever re-released, for example to fix a typo in the 20060531 schema, then the same undated schema URI ("…/av/rcs-event-v2.xsd") would automatically be updated to link to the updated version 2 schema for the "rcs-event" namespace.

Form 3 of the undated schema URI is always linked to the most recent release of the schema that represents the highest version of the namespace that has been published. For example, on June 25, 2002 (20020625), the undated schema URI "…/av/rcs-event.xsd" was linked to the schema that is otherwise known as "…/av/rcs-event-v1-20020625.xsd". However, on May 31, 2006 (20060531), that same undated schema URI was linked to the schema that is otherwise known as "…/av/rcs-event-v2-20060531.xsd".

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

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

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

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

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

The `version` attribute provides self-identification of the namespace version and release date of the schema itself. For example, within the original schema released for the "rcs-event" namespace (…/rcs-event-v2-20020625.xsd), the `<schema>` root element contains the following attribute: `version="2-20020625"`.

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

```
<?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.xsd
   urn:schemas-upnp-org:metadata-1-0/upnp/
    http://www.upnp.org/schemas/av/upnp.xsd">
   <item id="18" parentID="13" restricted="0">
      ...
   </item>
</DIDL-Lite>
```

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 below.

1.6.1 Vendor-defined Action Names

Vendor-defined action names MUST begin with "X_". Additionally, it SHOULD be followed by an ICANN assigned domain name owned by the vendor followed by the underscore character ("_"). It MUST then be followed by the vendor-assigned action name. The vendor-assigned action name MUST NOT contain a hyphen character ("-", 2D Hex in UTF-8) nor a hash character ("#", 23 Hex in UTF-8). Vendor-assigned action names are case sensitive. The first character of the name MUST be a US-ASCII letter ("A"-"Z", "a"-"z"), US-ASCII digit ("0"-"9"), an underscore ("_"), or a non-experimental Unicode letter or digit greater than U+007F. Succeeding characters MUST be a US-ASCII letter ("A"-"Z", "a"-"z"), US-ASCII digit ("0"-"9"), an underscore ("_"), a period ("."), a Unicode combiningchar, an extender, or a non-experimental Unicode letter or digit greater than U+007F. The first three letters MUST NOT be "XML" in any combination of case.

1.6.2 Vendor-defined State Variable Names

Vendor-defined state variable names MUST begin with "X_". Additionally, it SHOULD be followed by an ICANN assigned domain name owned by the vendor, followed by the underscore character ("_"). It MUST then be followed by the vendor-assigned state variable name. The vendor-assigned state variable name MUST NOT contain a hyphen character ("-", 2D Hex in UTF-8). Vendor-assigned action names are case sensitive. The first character of the name MUST be a US-ASCII letter ("A"-"Z", "a"-"z"), US-ASCII digit ("0"-"9"), an underscore ("_"), or a non-experimental Unicode letter or digit greater than U+007F. Succeeding characters MUST be a US-ASCII letter ("A"-"Z", "a"-"z"), US-ASCII digit ("0"-"9"), an underscore ("_"), a period ("."), a Unicode combiningchar, an extender, or a non-experimental Unicode letter or digit greater than U+007F. The first three letters MUST NOT be "XML" in any combination of case.

1.6.3 Vendor-defined XML Elements and attributes

UPnP vendors MAY add non-standard elements and attributes to a UPnP standard XML document, such as a device or service description. Each addition MUST be scoped by a vendor-owned XML namespace. Arbitrary XML MUST be enclosed in an element that begins with "X_," and this element MUST be a sub element of a standard complex type. Non-standard attributes MAY be added to standard elements provided these attributes are scoped by a vendor-owned XML namespace and begin with "X_".

1.6.4 Vendor-defined Property Names

UPnP vendors MAY add non-standard properties to the ContentDirectory service. Each property addition MUST be scoped by a vendor-owned namespace. The vendor-assigned property name MUST NOT contain a hyphen character ("-", 2D Hex in UTF-8). Vendor-assigned property names are case sensitive. The first character of the name MUST be a US-ASCII letter ("A"-"Z", "a"-"z"), US-ASCII digit ("0"-"9"), an underscore ("_"), or a non-experimental Unicode letter or digit greater than U+007F. Succeeding characters MUST be a US-ASCII letter ("A"-"Z", "a"-"z"), US-ASCII digit ("0"-"9"), an underscore ("_"), a period ("."), a Unicode combiningchar, an extender, or a non-experimental Unicode letter or digit greater than U+007F. The first three letters MUST NOT be "XML" in any combination of case.

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:

[ATRS-XSD] – *XML Schema for RenderingControl AllowedTransformSettings*, UPnP Forum, December 31, 2010.
Available at: http://www.upnp.org/schemas/av/AllowedTransformSettings-v1-20101231.xsd.
Latest version available at: http://www.upnp.org/schemas/av/AllowedTransformSettings.xsd.

[AVARCH] – *AVArchitecture:2*, UPnP Forum, December 31, 2010.
Available at: http://www.upnp.org/specs/av/UPnP-av-AVArchitecture-v2-20101231.pdf.
Latest version available at: http://www.upnp.org/specs/av/UPnP-av-AVArchitecture-v2.pdf.

[AVDT] – *AV Datastructure Template:1*, UPnP Forum, September 30, 2008.
Available at: http://www.upnp.org/specs/av/UPnP-av-AVDataStructureTemplate-v1-20080930.pdf.
Latest version available at: http://www.upnp.org/specs/av/UPnP-av-AVDataStructureTemplate-v1.pdf.

[AVDT-XSD] – *XML Schema for UPnP AV Datastructure Template:1*, UPnP Forum, September 30, 2008.
Available at: http://www.upnp.org/schemas/av/avdt-v1-20080930.xsd.
Latest version available at: http://www.upnp.org/schemas/av/avdt.xsd.

[AV-XSD] – *XML Schema for UPnP AV Common XML Data Types*, UPnP Forum, December 31, 2010.
Available at: http://www.upnp.org/schemas/av/av-v3-20101231.xsd.
Latest version available at: http://www.upnp.org/schemas/av/av.xsd.

[AVS-XSD] – *XML Schema for UPnP AV Common XML Structures*, UPnP Forum, December 31, 2010.
Available at: http://www.upnp.org/schemas/av/avs-v3-20101231.xsd.
Latest version available at: http://www.upnp.org/schemas/av/avs.xsd.

[AVT] – *AVTransport:3*, UPnP Forum, December 31, 2010.
Available at: http://www.upnp.org/specs/av/UPnP-av-AVTransport-v3-Service-20101231.pdf.
Latest version available at: http://www.upnp.org/specs/av/UPnP-av-AVTransport-v3-Service.pdf.

[AVT-EVENT-XSD] – *XML Schema for AVTransport LastChange Eventing*, UPnP Forum, September 30, 2008.
Available at: http://www.upnp.org/schemas/av/avt-event-v2-20080930.xsd.
Latest version available at: http://www.upnp.org/schemas/av/avt-event.xsd.

[CDS] – *ContentDirectory:4*, UPnP Forum, December 31, 2010.
Available at: http://www.upnp.org/specs/av/UPnP-av-ContentDirectory-v4-Service-20101231.pdf.
Latest version available at: http://www.upnp.org/specs/av/UPnP-av-ContentDirectory-v4-Service.pdf.

[CDS-EVENT-XSD] – *XML Schema for ContentDirectory LastChange Eventing*, UPnP Forum, September 30, 2008.
Available at: http://www.upnp.org/schemas/av/cds-event-v1-20080930.xsd.
Latest version available at: http://www.upnp.org/schemas/av/cds-event.xsd.

[CM] – *ConnectionManager:3*, UPnP Forum, December 31, 2010.
Available at: http://www.upnp.org/specs/av/UPnP-av-ConnectionManager-v3-Service-20101231.pdf.
Latest version available at: http://www.upnp.org/specs/av/UPnP-av-ConnectionManager-v3-Service.pdf.

[CM-DCIU-XSD] – *XML Schema for ConnectionManager DeviceClockInfoUpdates*, UPnP Forum, December 31, 2010.
Available at: http://www.upnp.org/schemas/av/cm-deviceClockInfoUpdates-v1-20101231.xsd.
Latest version available at: http://www.upnp.org/schemas/av/cm-deviceClockInfoUpdates.xsd.

[CM-FTRLST-XSD] – *XML Schema for ConnectionManager Features*, UPnP Forum, December 31, 2010.
Available at: http://www.upnp.org/schemas/av/cm-featureList-v1-20101231.xsd.
Latest version available at: http://www.upnp.org/schemas/av/cm-featureList.xsd.

[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, October 15, 2008.
Available at: http://www.upnp.org/specs/arch/UPnP-arch-DeviceArchitecture-v1.0-20081015.pdf.
Latest version available at: http://www.upnp.org/specs/arch/UPnP-arch-DeviceArchitecture-v1.0.pdf.

[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 Structure and Metadata (DIDL-Lite)*, UPnP Forum, December 31, 2010.
Available at: http://www.upnp.org/schemas/av/didl-lite-v3-20101231.xsd.
Latest version available at: http://www.upnp.org/schemas/av/didl-lite.xsd.

[DMO-XSD] – *XML Schema for ContentDirectory DeviceMode*, UPnP Forum, December 31, 2010.
Available at: http://www.upnp.org/schemas/av/dmo-v1-20101231.xsd.
Latest version available at: http://www.upnp.org/schemas/av/dmo.xsd.

[DMOR-XSD] – *XML Schema for ContentDirectory DeviceModeRequest*, UPnP Forum, December 31, 2010.
Available at: http://www.upnp.org/schemas/av/dmor-v1-20101231.xsd.
Latest version available at: http://www.upnp.org/schemas/av/dmor.xsd.

[DMOS-XSD] – *XML Schema for ContentDirectory DeviceModeStatus*, UPnP Forum, December 31, 2010.
Available at: http://www.upnp.org/schemas/av/dmos-v1-20101231.xsd.
Latest version available at: http://www.upnp.org/schemas/av/dmos.xsd.

[DP] – *DeviceProtection:1*, UPnP Forum, December 31, 2010.
Available at: http://www.upnp.org/specs/gw/UPnP-gw-DeviceProtection-v1-Service-20101231.pdf.
Latest version available at: http://www.upnp.org/specs/gw/UPnP-gw-DeviceProtection-v1-Service.pdf.

[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.

[IEEE-802.1AS] – *IEEE P802.1AS™ (Draft 7.0) - Timing and Synchronization for Time-Sensitive Applications in Bridged Local Area Networks*, Institute of Electrical and Electronics Engineers, March 23, 2010.
Available at: http://www.ieee802.org/1/pages/802.1as.html.

[IEEE-1733] – *IEEE-P1733™ (Draft 2.2) – Audio Video Bridge Layer 3 Transport Protocol*, International Institute of Electrical and Electronics Engineers, April 20, 2009.

Available at: http://grouper.ieee.org/groups/1733.

[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:3*, UPnP Forum, December 31, 2010.
Available at: http://www.upnp.org/specs/av/UPnP-av-MediaRenderer-v3-Device-20101231.pdf.
Latest version available at: http://www.upnp.org/specs/av/UPnP-AV-MediaRenderer-v3-Device.pdf.

[MS] – *MediaServer:4*, UPnP Forum, December 31, 2010.
Available at: http://www.upnp.org/specs/av/UPnP-av-MediaServer-v4-Device-20101231.pdf.
Latest version available at: http://www.upnp.org/specs/av/UPnP-AV-MediaServer-v4-Device.pdf.

[PI-XSD] – *XML Schema for ContentDirectory PermissionsInfo*, UPnP Forum, December 31, 2010.
Available at: http://www.upnp.org/schemas/av/pi-v1-20101231.xsd.
Latest version available at: http://www.upnp.org/schemas/av/pi.xsd.

[RCS] – *RenderingControl:3*, UPnP Forum, December 31, 2010.
Available at: http://www.upnp.org/specs/av/UPnP-av-RenderingControl-v3-Service-20101231.pdf.
Latest version available at: http://www.upnp.org/specs/av/UPnP-av-RenderingControl-v3-Service.pdf.

[RCS-EVENT-XSD] –*XML Schema for RenderingControl LastChange Eventing*, UPnP Forum, December 31, 2010.
Available at: http://www.upnp.org/schemas/av/rcs-event-v3-20101231.xsd.
Latest version available at: http://www.upnp.org/schemas/av/rcs-event.xsd.

[RFC 1305] – *IETF RFC 1305, Network Time Protocol (Version 3) Specification, Implementation and Analysis*, David L. Mills, March 1992.
Available at: http://www.ietf.org/rfc/rfc1305.txt.

[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 2030] – *IETF RFC 2030, Simple Network Time Protocol (SNTP) Version 4 for IPv4, IPv6 and OS*, D Mills, October 1996.
Available at: http://www.ietf.org/rfc/rfc2030.txt.

[RFC 2045] – *IETF RFC 2045, Multipurpose Internet Mail Extensions (MIME) Part 1:Format of Internet Message Bodies*, N. Freed, N. Borenstein, November 1996.
Available at: http://www.ietf.org/rfc/rfc2045.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.

[RII-XSD] – *XML Schema for ConnectionManager RendererInfo*, UPnP Forum, December 31, 2010.
Available at: http://www.upnp.org/schemas/av/rii-v1-20101231.xsd.
Latest version available at: http://www.upnp.org/schemas/av/rii.xsd.

[RPL-XSD] – *XML Schema for AVTransport PlaylistInfo*, UPnP Forum, December 31, 2010.

Available at: http://www.upnp.org/schemas/av/rpl-v1-20101231.xsd.
Latest version available at: http://www.upnp.org/schemas/av/rpl.xsd.

[RTP] – *IETF RFC 3550, RTP: A Transport Protocol for Real-Time Applications*, H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson, July 2003.
Available at: http://www.ietf.org/rfc/rfc3550.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:2*, UPnP Forum, December 31, 2010.
Available at: http://www.upnp.org/specs/av/UPnP-av-ScheduledRecording-v2-Service-20101231.pdf.
Latest version available at: http://www.upnp.org/specs/av/UPnP-av-ScheduledRecording-v2-Service.pdf.

[SRS-XSD] – *XML Schema for ScheduledRecording Metadata and Structure*, UPnP Forum, September 30, 2008.
Available at: http://www.upnp.org/schemas/av/srs-v2-20080930.xsd.
Latest version available at: http://www.upnp.org/schemas/av/srs.xsd.

[SRS-EVENT-XSD] – *XML Schema for ScheduledRecording LastChange Eventing*, UPnP Forum, September 30, 2008.
Available at: http://www.upnp.org/schemas/av/srs-event-v1-20080930.xsd.
Latest version available at: http://www.upnp.org/schemas/av/srs-event.xsd.

[TRS-XSD] – *XML Schema for RenderingControl TransformSettings*, UPnP Forum, December 31, 2010.
Available at: http://www.upnp.org/schemas/av/TransformSettings-v1-20101231.xsd.
Latest version available at: http://www.upnp.org/schemas/av/TransformSettings.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.

[UNITCONVERSION] – *Unit conversion.org website, definitions of more than 2100 units in 78 categories*.
Available at: http://www.unitconversion.org.

[UPNP-XSD] – *XML Schema for ContentDirectory Metadata*, UPnP Forum, December 31, 2010.
Available at: http://www.upnp.org/schemas/av/upnp-v4-20101231.xsd.
Latest version available at: http://www.upnp.org/schemas/av/upnp.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.

[UUID] – *IETF RFC 4122, A Universally Unique Identifier (UUID) URN Namespace*, P. Leach, Microsoft, M. Mealling, Refactored Networks LLC, R. Salz, DataPower Technology, Inc., July 2005.
Available at: http://www.ietf.org/rfc/rfc4122.txt.

[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.

[XPATH20] – *XML Path Language (XPath) 2.0*. Anders Berglund, Scott Boag, Don Chamberlin, Mary F. Fernandez, Michael Kay, Jonathan Robie, Jerome Simeon. W3C Recommendation, 21 November 2006.
Available at: http://www.w3.org/TR/xpath20.

[XQUERY10] – *XQuery 1.0 An XML Query Language*. W3C Recommendation, 23 January 2007.
Available at: http://www.w3.org/TR/2007/REC-xquery-20070123.

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:3*

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**	See Section 2.2.1		
PresetNameList	*R*	**string**	CSV[2] (**string**) See Section 2.2.2		
Brightness	*CR*[3]	**ui2**	See Table 2-2 and Section 2.2.3		
Contrast	*CR*[3]	**ui2**	See Table 2-3 and Section 2.2.4		
Sharpness	*CR*[3]	**ui2**	See Table 2-4 and Section 2.2.5		
RedVideoGain	*CR*[3]	**ui2**	See Table 2-5 and Section 2.2.6		
GreenVideoGain	*CR*[3]	**ui2**	See Table 2-6 and Section 2.2.7		
BlueVideoGain	*CR*[3]	**ui2**	See Table 2-7 and Section 2.2.8		
RedVideoBlackLevel	*CR*[3]	**ui2**	See Table 2-8 and Section 2.2.9		
GreenVideoBlackLevel	*CR*[3]	**ui2**	See Table 2-9 and Section 2.2.10		
BlueVideoBlackLevel	*CR*[3]	**ui2**	See Table 2-10 and Section 2.2.11		
ColorTemperature	*CR*[3]	**ui2**	See Table 2-11 and Section 2.2.12		
HorizontalKeystone	*CR*[3]	**i2**	See Table 2-12 and Section 2.2.13		
VerticalKeystone	*CR*[3]	**i2**	See Table 2-13 and Section 2.2.14		
Mute	*CR*[3]	**boolean**	See Section 2.2.15		
Volume[4]	*CR*[3]	**ui2**	See Table 2-14 and Section 2.2.16		

Variable Name	R/O[1]	Data Type	Allowed Value	Default Value	Eng. Units
VolumeDB	CR[3]	i2	See Table 2-15 and Section 2.2.17		1/256 dB
Loudness	CR[3]	boolean	See Section 2.2.18		
AllowedTransformSettings	CR[3]	string	See Section 2.2.19		
TransformSettings	CR[3]	string	See Section 2.2.20		
AllowedDefaultTransformSettings	CR[3]	string	See Section 2.2.21		
DefaultTransformSettings	CR[3]	string	See Section 2.2.22		
A_ARG_TYPE_Channel	R	string	See Table 2-16 and Section 2.2.23		
A_ARG_TYPE_InstanceID	R	ui4	See Section 2.2.24		
A_ARG_TYPE_PresetName	R	string	See Table 2-17 and Section 2.2.25		
A_ARG_TYPE_DeviceUDN	CR[3]	string	See Section 2.2.26		
A_ARG_TYPE_ServiceType	CR[3]	string	See Section 2.2.27		
A_ARG_TYPE_ServiceID	CR[3]	string	See Section 2.2.28		
A_ARG_TYPE_ StateVariableValuePairs	CR[3]	string	See Section 2.2.29		
A_ARG_TYPE_ StateVariableList	CR[3]	string	CSV (string) See Section 2.2.30		
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] See section of the relevant state variable for conditions under which implementation of this state variable is REQUIRED.

[4] 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*

	Value	R/O
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
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	Description
"*Master*"	*R*	Master volume
"*LF*"	*O*	Left Front
"*RF*"	*O*	Right Front
"*CF*"	*O*	Center Front
"*LFE*"	*O*	Low Frequency Enhancement
"*LS*"	*O*	Left Surround
"*RS*"	*O*	Right Surround
"*LFC*"	*O*	Left of Center [in front]
"*RFC*"	*O*	Right of Center [in front]
"*SD*"	*O*	Surround [rear]
"*SL*"	*O*	Side Left [left wall]
"*SR*"	*O*	Side Right [right wall]
"*T*"	*O*	Top [overhead]
"*B*"	*O*	Bottom
"*BC*"	*O*	Back Center
"*BL*"	*O*	Back Left
"*BR*"	*O*	Back Right
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 REQUIRED 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>
   <InstanceID val="2">
      <AllowedTransformSettings val="
         <?xml version="1.0" encoding="UTF-8"?>
         <TransformList
            xmlns="urn:schemas-upnp-org:av:AllowedTransformSettings"
            xmlns:xsi"http://www.w3.org/2001/XMLSchema-instance"
            xsi:schemaLocation=
               "urn:schemas-upnp-org:av:AllowedTransformSettings
      http://www.upnp.org/schemas/av/AllowedTransformSettings.xsd">
            <transform name="Rotation">
               <allowedValueRange inactiveValue="0">
                  <minimum>0</minimum>
                  <maximum>270</maximum>
                  <step>90</step>
               </allowedValueRange>
            </transform>
            <transform name="Zoom">
               <allowedValueRange unit="pct" info="Zoom factor in
percent">
                  <minimum>50</minimum>
                  <maximum>400</maximum>
                  <step>1</step>
               </allowedValueRange>
               <allowedValueList info="Pre-defined Zoom values"
inactiveValue="None">
                  <allowedValue>None</allowedValue>
                  <allowedValue>N/A</allowedValue>
                  <allowedValue>Zoom 1</allowedValue>
                  <allowedValue>Zoom 2</allowedValue>
               </allowedValueList>
            </transform>
            <!-- ... remaining transforms allowed for this instance -->
         </TransformList>"
      />
      <TransformSettings val="
         <?xml version="1.0" encoding="UTF-8"?>
         <TransformSettings xmlns=
            "urn:schemas-upnp-org:av:TransformSettings"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:schemaLocation="urn:schemas-upnp-
org:av:TransformSettings
           http://www.upnp.org/schemas/av/TransformSettings.xsd">
            <transform name="Rotation">
               <value>0</value>
            </transform>
            <transform name="Zoom">
               <value>100</value>
               <value index="1">None</value>
            </transform>
            <!-- ... remaining transform settings for this instance -->
         </TransformSettings>"
      />
      ...
   </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. For instance 2 there are 2 transforms, *Rotation* and *Zoom*. The *Rotation* transform has the set of allowed values of [0, 90, 180, 270] and has currently the value of 0 (no rotation). The *Zoom* transform has the allowed value range of 50% to 400% and has been set to 100%. Additionally, the *Zoom* transform has a number of pre-defined vendor-specific allowed values. For a list of pre-defined transforms see Appendix A.

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, the *Brightness* of instance 1 changes from 54 to 35, then to 11, and a transform has been performed on instance 2 such that the *TransformSettings* are modified, *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>
   <InstanceID val="2">
      <TransformSettings val="
      <!-- ... the following text needs to be XML escaped -->
         <?xml version="1.0" encoding="UTF-8"?>
         <TransformSettings xmlns=
            "urn:schemas-upnp-org:av:TransformSettings"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:schemaLocation="urn:schemas-upnp-
org:av:TransformSettings
           http://www.upnp.org/schemas/av/TransformSettings.xsd">
            <transform name="Rotation">
               <value>90</value>
            </transform>
            <transform name="Zoom">
               <value>100</value>
            </transform>
         </TransformSettings>"
      />
   </InstanceID>
</Event>
```

The *LastChange* state variable is evented using the standard UPnP event mechanism. All other state variables , except the state variables *AllowedDefaultTransformSettings* and *DefaultTransformSettings*, are indirectly evented via the *LastChange* state variable event. When a new instance is created by invoking the *ConnectionManager::PrepareForConnection()* action, the *LastChange* state variable MUST incorporate the new instance and thus be evented with the initial values. 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.

Note that for all values of state variables that contain XML, the content MUST be escaped before inserting it inside *LastChange*.

Note when a new instance is created, the initial values for transforms SHOULD be equal to the values in the *DefaultTransformSettings* state variable, or the transform's inactive value when no default value is established.

2.2.2 *PresetNameList*

This REQUIRED 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 CONDITIONALLY REQUIRED state variable MUST be implemented if the *GetBrightness()* or *SetBrightness()* actions are implemented. The 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 CONDITIONALLY REQUIRED state variable MUST be implemented if the *GetContrast()* or *SetContrast()* actions are implemented. The 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 CONDITIONALLY REQUIRED state variable MUST be implemented if the *GetSharpness()* or *SetSharpness()* actions are implemented. The 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 CONDITIONALLY REQUIRED state variable MUST be implemented if the *GetRedVideoGain()* or *SetRedVideoGain()* actions are implemented. The 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 2.2.31.1, "*xxxVideoGain* and *xxxVideoBlackLevel*" for additional information.

2.2.7 *GreenVideoGain*

This CONDITIONALLY REQUIRED state variable MUST be implemented if the *GetGreenVideoGain()* or *SetGreenVideoGain()* actions are implemented. The 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 2.2.31.1, "*xxxVideoGain* and *xxxVideoBlackLevel*" for additional information.

2.2.8 *BlueVideoGain*

This CONDITIONALLY REQUIRED state variable MUST be implemented if the *GetBlueVideoGain()* or *SetBlueVideoGain()* actions are implemented. The 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 2.2.31.1, "*xxxVideoGain* and *xxxVideoBlackLevel*" for additional information.

2.2.9 *RedVideoBlackLevel*

This CONDITIONALLY REQUIRED state variable MUST be implemented if the *GetRedVideoBlackLevel()* or *SetRedVideoBlackLevel()* actions are implemented. The 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 2.2.31.1, "*xxxVideoGain* and *xxxVideoBlackLevel*" for additional information.

2.2.10 *GreenVideoBlackLevel*

This CONDITIONALLY REQUIRED state variable MUST be implemented if the *GetGreenVideoBlackLevel()* or *SetGreenVideoBlackLevel()* actions are implemented. The 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 2.2.31.1, "*xxxVideoGain* and *xxxVideoBlackLevel*" for additional information.

2.2.11 *BlueVideoBlackLevel*

This CONDITIONALLY REQUIRED state variable MUST be implemented if the *GetBlueVideoBlackLevel()* or *SetBlueVideoBlackLevel()* actions are implemented. The 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 2.2.31.1, "*xxxVideoGain* and *xxxVideoBlackLevel*" for additional information.

2.2.12 *ColorTemperature*

This CONDITIONALLY REQUIRED state variable MUST be implemented if the *GetColorTemperature()* or *SetColorTemperature()* actions are implemented. The 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 CONDITIONALLY REQUIRED state variable MUST be implemented if the *GetHorizontalKeystone()* or *SetHorizontalKeystone()* actions are implemented. The 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.

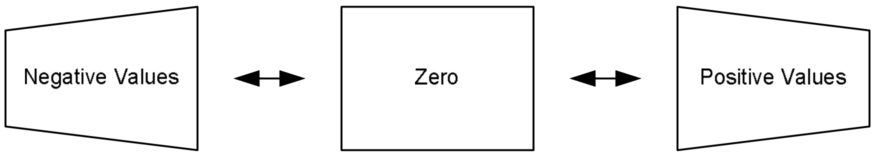

Figure 1: Horizontal Keystone

2.2.14 *VerticalKeystone*

This CONDITIONALLY REQUIRED state variable MUST be implemented if the *GetVerticalKeystone()* or *SetVerticalKeystone()* actions are implemented. The 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.

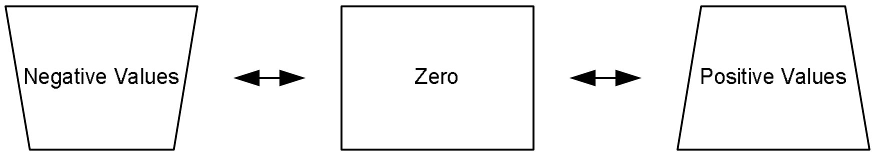

Figure 2: Vertical Keystone

2.2.15 *Mute*

This CONDITIONALLY REQUIRED state variable MUST be implemented if the *GetMute()* or *SetMute()* actions are implemented. The 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 CONDITIONALLY REQUIRED state variable MUST be implemented if the *GetVolume()* or *SetVolume()* actions are implemented. The 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 CONDITIONALLY REQUIRED state variable MUST be implemented if the *GetVolumeDB()*, *SetVolumeDB()* or *GetVolumeDBRange()* actions are implemented. The 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 CONDITIONALLY REQUIRED state variable MUST be implemented if the *GetLoudness()* or *SetLoudness()* actions are implemented. The 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 *AllowedTransformSettings*

This CONDITIONALLY REQUIRED state variable MUST be implemented if the *GetAllowedTransforms()* or *GetAllAvailableTransforms()* actions are implemented. The state variable provides information about the currently allowed set of transforms that can be applied on the associated device.

The *AllowedTransformSettings* state variable is a string that contains an *AllowedTransformSettings XML Document*. The state variable is evented on a per instance basis conveyed by the *LastChange* state variable.

When an instance is created (by the *PrepareForConnection()* action) the initial value of this state variable MUST convey all transforms that the implementation supports when no content is associated with the rendering control instance. Prior to an initial media item being downloaded to a virtual renderer instance using the *SetAVTransportURI()* action, an empty allowed transform state variable MAY be reported. However, in cases where media-dependent transforms may be applied such as the *Brightness* transform, the implementation MAY report these transforms as immediately available after instance creation. The choice of this behavior is implementation dependent. When a change occurs to the *AVTransportURI* or *CurrentTrackURI* AVTransport service state variables, a new media item is bound to an instance. The renderer MUST determine if updates to the current allowed transform list are required. If changes to the allowed transform list are detected, a complete new transform list SHOULD be evented as quickly as possible.

When any allowed transform or associated allowed values are modified then the new evented value of this state variable MUST reflect the entire set of allowed transforms for this instance.

See the *AllowedTransformSettings* schema [AVS-XSD] for details.

The following example shows a generalized “template” for the format of the *AllowedTransformSettings XML Document*. The example shows the elements that need to be filled out by individual implementations in the *vendor* character style.

```
<?xml version="1.0" encoding="UTF-8"?>
<TransformList
 xmlns="urn:schemas-upnp-org:av:AllowedTransformSettings"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="urn:schemas-upnp-org:av:AllowedTransformSettings
      http://www.upnp.org/schemas/av/AllowedTransformSettings.xsd">
   <transform name="Name of transform" shared="1|0">
      <friendlyName>Friendly name of transform</friendlyName>
      <allowedValueRange
         unit="unit of the transform"
         scale="scale indication of the transform"
         inactiveValue="value of the transform that does not result in
any effect on the (original) output"
         >
         <minimum>minimum value</minimum>
         <maximum>maximum value</maximum>
         <step>increment value</step>
      </allowedValueRange>
   </transform>
   <transform name="Name of transform" shared="1|0">
      <friendlyName>Friendly name of transform</friendlyName>
      <allowedValueList
         unit="unit of the transform"
         scale="scale indication of the transform"
         inactiveValue="value of the transform that does not result in
any effect on the (original) output"
      >
         <allowedValue>enumerated value</allowedValue>
         <!-- other allowed values go here -->
      </allowedValueList>
   </transform>
   <!-- other transforms go here -->
</TransformList>
```

xml

REQUIRED for all XML documents. Case sensitive.

TransformList

REQUIRED. Must have "urn:schemas-upnp-org:av:AllowedTransformSettings" as the value for the xmlns attribute; this references the UPnP AV Working Committee Datastructure Template Schema.

transform

OPTIONAL. xsd:string. Identifies the ***transform*** and its corresponding allowed value set.

@name

REQUIRED. xsd:string. Identifies the name of the **transform** that is described within this element. The value of this attribute is case sensitive.

@shared

REQUIRED. xsd:boolean. A value of "*1*" indicates that the **transform** is shared. A shared transform acts on the post-mix instance of all virtual renderer instances. When the **transform** is identified as shared, the **transform** changes MUST be applied on *InstanceID* = 0.

friendlyName

OPTIONAL. xsd:string. A short user friendly name for the **transform**. Only one **friendlyName** element can be present per transform.

allowedValueList

OPTIONAL. Enumerates a set of values that are allowed for this element. If the `allowedValueList` element is not present then the `allowedValueRange` element MUST be present. Contains the following child elements and attributes:

allowedValue

REQUIRED. xsd:anyType. Identifies one of the values that are allowed for this **element**. Legal values are typically defined by the UPnP Forum AV Working Committee. However, vendors MAY add vendor-specific allowed values.

@unit

OPTIONAL. xsd:string. Indicates the unit of measure in which the transform settings are expressed. The units defined in the table below MUST be used by the implementations when a transforms uses an applicable measure. Only one `unit` attribute instance can be present per set of allowed values. For other unit definitions see [UNITCONVERSION]

Table 2-18: Allowed values for the `unit` attribute.

Unit	Measure	Comments (Usages)
lv	Brightness in Cd/m^2	Display brightness.
dB	Sound in decibels	Volume adjustments.
deg	Angle	Rotation, Tilt adjustments.
dur	Duration in Years/Days/Hours/Min/Sec	PnYnMnDTnHnMnS format. Device sleep timers. Inactivity timer. Energy saving.
C	Temperature in Celsius	Equipment, Room temperatures.
K	Temperature in Kelvin	Color temperature.
pct	Percentage	Relative display positioning.
px	Graphical in pixel	Absolute positioning on display.
pt	Graphical in point	OSD font sizes.
Hz	Frequency in Hertz	Display response times. Refresh rates. Audio spectrum.
kHz	Frequency in kilohertz	Display response times. Refresh rates. Audio spectrum.
MHz	Frequency in megahertz	Display response times. Refresh rates. Audio spectrum.
GHz	Frequency in gigahertz	Display response times. Refresh rates. Audio spectrum.
s	Time in seconds	Photo hold times. Transition effects speed. Screen Savers.
ms	Time in milliseconds	Photo hold times. Transition effects speed. Screen Savers.
m	Length in meters	Room dimensions. Distances between speakers. Distance to screen.
cm	Length in centimeters	Room dimensions. Distances between speakers. Distance to screen.
mm	Length in millimeters	Room dimensions. Distances between speakers. Distance to screen.
W	Power in Watts	Power.
kW-h	Power consumption in kilowatt-hours	Power usage over time.
Vendor-defined		Vendors SHOULD use standard defined

		units like unit in the SI system.

@scale

OPTIONAL. xsd:string. Indicates the scale of the transform settings of this transform. Only one **scale** attribute instance can be present per set of allowed values.

Table 2-19: Allowed values for the `scale` attribute.

Scale	Description
LINEAR	Scale is linear spaced
LOGARITHMIC	Scale is logarithmic spaced
Vendor-defined	

@inactiveValue

OPTIONAL. xsd:string. Indicates the value of the transform which, when set, results in the same output as the original rendered output.

@info

OPTIONAL. xsd:string. Indicates an informative descriptive name for this set of allowed values.

allowedValueRange

OPTIONAL. Defines a range and resolution for a set of numeric values that are allowed for this **element.** If the `allowedValueRange` element is not present then the `allowedValueList` element MUST be present. Contains the following child elements and attributes:

minimum

REQUIRED. xsd:string. Single numeric value. Inclusive lower bound. MUST be less than `maximum`.

Note: Care must be taken when dealing with floating-point values so that conversion and/or rounding errors do not cause inaccurate comparison operations.

maximum

REQUIRED. xsd:string. Single numeric value. Inclusive upper bound. MUST be greater than `minimum`.

Note: Care must be taken when dealing with floating-point values so that conversion and/or rounding errors do not cause inaccurate comparison operations.

step

OPTIONAL. xsd:string. Single positive numeric value. Indicates the numeric difference between adjacent valid values within the `allowedValueRange`. The value of `step` MUST divide the inclusive range from `minimum` to `maximum` into an integral number of equal parts. In other words, `maximum` = `minimum` + N*`step` where N is a positive integer. When `step` is omitted and the data type of the **element** is an integer, the default value of **step** is 1. Otherwise, if `step` is omitted, all values within the inclusive range from `minimum` to `maximum` MUST be supported.

Note: Care must be taken when dealing with floating-point values so that conversion and/or rounding errors do not cause inaccurate comparison operations.

@unit

See `unit` description from the allowed value list.

@scale

See `scale` description from the allowed value list.

@inactiveValue

See `inactiveValue` description from the allowed value list.

@info

See `info` description from the allowed value list.

`Transforms` that are specified with numeric values can be conveyed with `allowedValueRange` or with `allowedValueList`. This is a vendor specific choice. Numeric values can be specified either as a float or as an integer. This can be deducted by the precision of the `step` value when the `allowedValueRange` method is used.

Note that since the value of *AllowedTransformSettings* is XML, it needs to be properly escaped (using the normal XML rules: **[XML]** Section 2.4 Character Data and Markup) before embedding in a SOAP response message.

Note that for all values of state variables that contain XML, the content MUST be escaped before inserting it inside *LastChange*.

2.2.20 *TransformSettings*

This CONDITIONALLY REQUIRED state variable MUST be implemented if the *GetTransforms()* or *SetTransforms()* actions are implemented. The state variable provides information about the current values of the set of applicable transforms. The initial values for the transforms settings from a newly created instance are copied from the currently established default values.

Note: When one transform and its current value are changed, all transforms will be evented for that instance.

See the *TransformSettings* schema [AVS-XSD] for details.

The following example shows a generalized "template" for the format of the *TransformSettings XML Document*. The example shows elements that need to be filled out by individual implementations in the *vendor* character style.

```
<?xml version="1.0" encoding="UTF-8"?>
<TransformSettings
 xmlns="urn:schemas-upnp-org:av:TransformSettings"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="urn:schemas-upnp-org:av:TransformSettings
      http://www.upnp.org/schemas/av/TransformSettings.xsd">
   <transform name="Name of transform">
      <value>value<value>
      <value index="1">value-1<value>
   </transform>
   <!-- other transforms go here -->
</TransformSettings>
```

xml

REQUIRED for all XML documents. Case sensitive.

TransformSettings

REQUIRED. Must have "urn:schemas-upnp-org:av:TransformSettings" as the value for the xmlns attribute; this references the UPnP AV Working Committee Datastructure Template Schema. As long as the same xmlns is used, the data structure template MUST be backward compatible, i.e. usable by legacy implementations. This element contains a list of transforms settings. The transforms are designed orthogonal, so order of the transforms in this list is not important.

transform

OPTIONAL. xsd:string. Identifies the data structure type. Contains the following child element and attribute:

@name

REQUIRED. xsd:string. Identifies the name of the `transform` that is described within this element.

value

REQUIRED. xsd:string. The `value` element MUST have a value that is part of the current set of allowed values of the transform.

@index

Optional. xsd:string. This value indicates which set of allowed values of the transform is being used. The `@index` attribute MUST have a value that indexes the multiple allowed value lists or ranges indicated on the transform. When the index value is omitted the value is 0.

Note that since the value of *TransformSettings* is XML, it needs to be properly escaped (using the normal XML rules: **[XML]** Section 2.4 Character Data and Markup) before embedding in a SOAP response message.

Note that for all values of state variables that contain XML, the content MUST be escaped before inserting it inside *LastChange*.

2.2.21 *AllowedDefaultTransformSettings*

This CONDITIONALLY REQUIRED state variable MUST be implemented if the *GetAllowedDefaultTransformSettings()* actions is implemented. The state variable is introduced to provide type information for the *AllowedDefaultTransformSettings* argument in the *GetAllowedDefaultTransforms()* action. This state variable has the same definition as the *AllowedTransformSettings* state variable.

This state variable MUST be evented when one of the allowed default values is changed by the implementation.

Note that the *AllowedDefaultTransformSettings* state variable is NOT part of the *LastChange* state variable.

Note that the change of this state variable is induced from an out-of-band method and that the content of this state variable does not change often.

2.2.22 *DefaultTransformSettings*

This CONDITIONALLY REQUIRED state variable MUST be implemented if the *GetDefaultTransforms()* or *SetDefaultTransforms()* actions are implemented. The state variable is introduced to provide type information for the *CurrentDefaultTransformSettings* argument in the *GetDefaultTransforms()* action, and for the *DesiredDefaultTransformSettings* argument in the *SetDefaultTransforms()* action. This state variable has the same definition as the *TransformSettings* state variable (See section 2.2.20).

This state variable MUST be evented when one of the default values is changed by invoking the *SetDefaultTransforms()* action.

Note that the *DefaultTransformSettings* state variable is NOT part of the *LastChange* state variable.

2.2.23 *A_ARG_TYPE_Channel*

This REQUIRED 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 in Table 2-16.

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.

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.

2.2.24 *A_ARG_TYPE_InstanceID*

This REQUIRED 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.

2.2.25 *A_ARG_TYPE_PresetName*

This REQUIRED 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 2-20: Predefined Names of Some Common Presets

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

2.2.26 *A_ARG_TYPE_DeviceUDN*

This CONDITIONALLY REQUIRED state variable MUST be supported if the *SetStateVariables()* action is implemented. The state variable is introduced to provide type information for the *RenderingControlUDN* argument in this action. It is a **string** value containing the UDN of the device.

2.2.27 *A_ARG_TYPE_ServiceType*

This CONDITIONALLY REQUIRED state variable MUST be supported if the *SetStateVariables()* action is implemented. The state variable is introduced to provide type information for the *ServiceType* argument in this action. It is a **string** value containing the service type and version number of a service such as "RenderingControl:3".

2.2.28 *A_ARG_TYPE_ServiceID*

This CONDITIONALLY REQUIRED state variable MUST be supported if the *SetStateVariables()* action is implemented. The state variable is introduced to provide type information for the *ServiceId* argument in this action. It is a **string** value containing the service ID of a service.

2.2.29 *A_ARG_TYPE_StateVariableValuePairs*

This CONDITIONALLY REQUIRED state variable MUST be supported if the *GetStateVariables()* or *SetStateVariables()* action is implemented. The state variable is introduced to provide type information for the *StateVariableValuePairs* argument in these 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.

2.2.30 *A_ARG_TYPE_StateVariableList*

This CONDITIONALLY REQUIRED state variable MUST be supported if the *GetStateVariables()* or *SetStateVariables()* action is implemented. The state variable is introduced to provide type information for the *StateVariableList* argument in these 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.)

2.2.31 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.

2.2.31.1 *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*

2.2.31.2 *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.

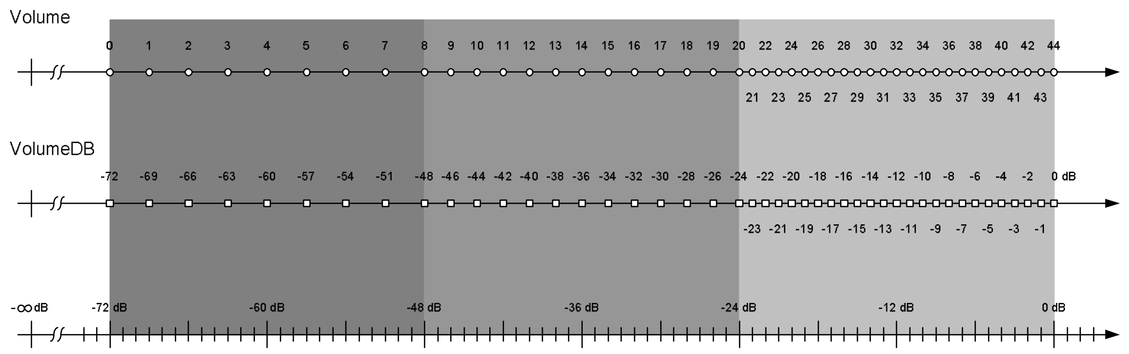

Figure 3: Relationship between *Volume* and *VolumeDB*

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 2-21: 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*			

Variable Name	Evented	Moderated Event	Max Event Rate[1]	Logical Combination	Min Delta per Event[2]
ColorTemperature	*NO*	*NO*			
HorizontalKeystone	*NO*	*NO*			
VerticalKeystone	*NO*	*NO*			
Mute	*NO*	*NO*			
Volume	*NO*	*NO*			
VolumeDB	*NO*	*NO*			
Loudness	*NO*	*NO*			
AllowedTransformSettings	*NO*	*NO*			
TransformSettings	*NO*	*NO*			
AllowedDefault TransformSettings	*YES*	*NO*			
DefaultTransformSettings	*YES*	*NO*			
A_ARG_TYPE_Channel	*NO*	*NO*			
A_ARG_TYPE_InstanceID	*NO*	*NO*			
A_ARG_TYPE_PresetName	*NO*	*NO*			
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 2-22: Actions

Name	R/O[1]	Control Point R/O[2]
ListPresets()	*R*	*O*
SelectPreset()	*R*	*O*
GetBrightness()	*O*	*O*
SetBrightness()	*O*	*O*
GetContrast()	*O*	*O*
SetContrast()	*O*	*O*
GetSharpness()	*O*	*O*
SetSharpness()	*O*	*O*
GetRedVideoGain()	*O*	*O*
SetRedVideoGain()	*O*	*O*
GetGreenVideoGain()	*O*	*O*
SetGreenVideoGain()	*O*	*O*
GetBlueVideoGain()	*O*	*O*

Name	R/O[1]	Control Point R/O[2]
SetBlueVideoGain()	*O*	*O*
GetRedVideoBlackLevel()	*O*	*O*
SetRedVideoBlackLevel()	*O*	*O*
GetGreenVideoBlackLevel()	*O*	*O*
SetGreenVideoBlackLevel()	*O*	*O*
GetBlueVideoBlackLevel()	*O*	*O*
SetBlueVideoBlackLevel()	*O*	*O*
GetColorTemperature()	*O*	*O*
SetColorTemperature()	*O*	*O*
GetHorizontalKeystone()	*O*	*O*
SetHorizontalKeystone()	*O*	*O*
GetVerticalKeystone()	*O*	*O*
SetVerticalKeystone()	*O*	*O*
GetMute()	*O*	*O*
SetMute()	*O*	*O*
GetVolume()	*O*	*O*
SetVolume()	*O*	*O*
GetVolumeDB()	*O*	*O*
SetVolumeDB()	*O*	*O*
GetVolumeDBRange()	*CR*[3]	*O*
GetLoudness()	*O*	*O*
SetLoudness()	*O*	*O*
GetStateVariables()	*O*	*O*
SetStateVariables()	*O*	*O*
GetAllowedTransforms()	*CR*[3]	*O*
GetTransforms()	*CR*[3]	*O*
SetTransforms()	*CR*[3]	*O*
GetAllowedDefaultTransforms()	*CR*[3]	*O*
GetDefaultTransforms()	*CR*[3]	*O*
SetDefaultTransforms()	*CR*[3]	*O*
GetAllAvailableTransforms()	*CR*[3]	*O*
Non-standard actions implemented by a UPnP vendor go here	*X*	*X*

[1] For a device this column indicates whether the action MUST be implemented or not, where *R* = REQUIRED, *O* = OPTIONAL, *CR* = CONDITIONALLY REQUIRED, *CO* = CONDITIONALLY OPTIONAL, *X* = Non-standard, add *-D* when deprecated (e.g., *R-D*, *O-D*).

[2] For a control point this column indicates whether a control point MUST be capable of invoking this action, where *R* = REQUIRED, *O* = OPTIONAL, *CR* = CONDITIONALLY REQUIRED, *CO* = CONDITIONALLY OPTIONAL, *X* = Non-standard, add *-D* when deprecated (e.g., *R-D*, *O-D*).

[3] See action description for conditions under which implementation of this action is REQUIRED.

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 REQUIRED 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 2-23: 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 2-24: 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 REQUIRED 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 2-20, "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.

Note: When the *FactoryDefaults* preset is specified, the current value of each transform with a specified default value MUST return to factory defaults as defined by the vendor.

Note: When the *InstallationDefaults* preset is specified, the current value of each transform with a specified default value MUST return to the current default value specified by the transform.

2.4.2.1 Arguments

Table 2-25: 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 2-26: 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 OPTIONAL action retrieves the current value of the *Brightness* state variable of the specified instance of this service.

Note: this functionality is also offered by the *Brightness* transform, see Appendix A.16.5.

2.4.3.1 Arguments

Table 2-27: 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 2-28: 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 OPTIONAL action sets the *Brightness* state variable of the specified instance of this service to the specified value.

Note: this functionality is also offered by the *Brightness* transform, see Appendix A.16.5.

2.4.4.1 Arguments

Table 2-29: 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 2-30: 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 OPTIONAL action retrieves the current value of the *Contrast* state variable of the specified instance of this service.

Note: this functionality is also offered by the *Contrast* transform, see Appendix A.16.7.

2.4.5.1 Arguments

Table 2-31: 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 2-32: 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 OPTIONAL action sets the *Contrast* state variable of the specified instance of this service to the specified value.

Note: this functionality is also offered by the *Contrast* transform, see Appendix A.16.7.

2.4.6.1 Arguments

Table 2-33: 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 2-34: 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 OPTIONAL action retrieves the current value of the *Sharpness* state variable of the specified instance of this service.

Note: this functionality is also offered by the *Sharpness* transform, see Appendix A.16.6.

2.4.7.1 Arguments

Table 2-35: 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 2-36: 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 OPTIONAL action sets the *Sharpness* state variable of the specified instance of this service to the specified value.

Note: this functionality is also offered by the *Sharpness* transform, see Appendix A.16.6.

2.4.8.1 Arguments

Table 2-37: 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 2-38: Error Codes for *SetSharpness()*

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

errorCode	errorDescription	Description
500-599	TBD	See UPnP Device Architecture section on Control.
600-699	TBD	See UPnP Device Architecture section on Control.
702	Invalid InstanceID	The specified *InstanceID* is invalid.

2.4.9 *GetRedVideoGain()*

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

Note: this functionality is also offered by the *RedVideoGain* transform, see Appendix A.16.8.

2.4.9.1 Arguments

Table 2-39: 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 2-40: 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 OPTIONAL action sets the *RedVideoGain* state variable of the specified instance of this service to the specified value.

Note: this functionality is also offered by the *RedVideoGain* transform, see Appendix A.16.12.

2.4.10.1 Arguments

Table 2-41: 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 2-42: 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 OPTIONAL action retrieves the current value of the *GreenVideoGain* state variable of the specified instance of this service.

Note: this functionality is also offered by the *GreenVideoGain* transform, see Appendix A.16.9.

2.4.11.1 Arguments

Table 2-43: 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 2-44: 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 OPTIONAL action sets the *GreenVideoGain* state variable of the specified instance of this service to the specified value.

Note: this functionality is also offered by the *GreenVideoGain* transform, see Appendix A.16.9.

2.4.12.1 Arguments

Table 2-45: 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 2-46: 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 OPTIONAL action retrieves the current value of the *BlueVideoGain* state variable of the specified instance of this service.

Note: this functionality is also offered by the *BlueVideoGain* transform, see Appendix A.16.10.

2.4.13.1 Arguments

Table 2-47: 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 2-48: 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.
600-699	TBD	See UPnP Device Architecture section on Control.
702	Invalid InstanceID	The specified *InstanceID* is invalid.

2.4.14 *SetBlueVideoGain()*

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

Note: this functionality is also offered by the *BlueVideoGain* transform, see Appendix A.16.10.

2.4.14.1 Arguments

Table 2-49: 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 2-50: 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 OPTIONAL action retrieves the current value of the *RedVideoBlackLevel* state variable of the specified instance of this service.

Note: this functionality is also offered by the *RedVideoBlackLevel* transform, see Appendix A.16.11.

2.4.15.1 Arguments

Table 2-51: 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 2-52: 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 OPTIONAL action sets the *RedVideoBlackLevel* state variable of the specified instance of this service to the specified value.

Note: this functionality is also offered by the *RedVideoBlackLevel* transform, see Appendix A.16.11.

2.4.16.1 Arguments

Table 2-53: 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 2-54: Error Codes for *SetRedVideoBlackLevel()*

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

errorCode	errorDescription	Description
500-599	TBD	See UPnP Device Architecture section on Control.
600-699	TBD	See UPnP Device Architecture section on Control.
702	Invalid InstanceID	The specified *InstanceID* is invalid.

2.4.17 *GetGreenVideoBlackLevel()*

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

Note: this functionality is also offered by the *GreenVideoBlackLevel* transform, see Appendix A.16.12.

2.4.17.1 Arguments

Table 2-55: 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 2-56: 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 OPTIONAL action sets the *GreenVideoBlackLevel* state variable of the specified instance of this service to the specified value.

Note: this functionality is also offered by the *GreenVideoBlackLevel* transform, see Appendix A.16.12.

2.4.18.1 Arguments

Table 2-57: 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 2-58: 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.
600-699	TBD	See UPnP Device Architecture section on Control.
702	Invalid InstanceID	The specified *InstanceID* is invalid.

2.4.19 *GetBlueVideoBlackLevel()*

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

Note: this functionality is also offered by the *BlueVideoBlackLevel* transform, see Appendix A.16.13.

2.4.19.1 Arguments

Table 2-59: 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 2-60: 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 OPTIONAL action sets the *BlueVideoBlackLevel* state variable of the specified instance of this service to the specified value.

Note: this functionality is also offered by the *BlueVideoBlackLevel* transform, see Appendix A.16.13.

2.4.20.1 Arguments

Table 2-61: 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 2-62: 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 OPTIONAL action retrieves the current value of the *ColorTemperature* state variable of the specified instance of this service.

Note: this functionality is also offered by the *ColorTemperature* transform, see Appendix A.16.14.

2.4.21.1 Arguments

Table 2-63: 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 2-64: 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 OPTIONAL action sets the *ColorTemperature* state variable of the specified instance of this service to the specified value.

Note: this functionality is also offered by the *ColorTemperature* transform, see Appendix A.16.14.

2.4.22.1 Arguments

Table 2-65: 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 2-66: 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 OPTIONAL action retrieves the current value of the *HorizontalKeystone* state variable of the specified instance of this service.

Note: this functionality is also offered by the *HorizontalKeystone* transform, see Appendix A.16.15.

2.4.23.1 Arguments

Table 2-67: 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 2-68: 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.
600-699	TBD	See UPnP Device Architecture section on Control.
702	Invalid InstanceID	The specified *InstanceID* is invalid.

2.4.24 *SetHorizontalKeystone()*

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

Note: this functionality is also offered by the *HorizontalKeystone* transform, see Appendix A.16.15.

2.4.24.1 Arguments

Table 2-69: 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 2-70: Error Codes for *SetHorizontalKeystone()*

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

errorCode	errorDescription	Description
500-599	TBD	See UPnP Device Architecture section on Control.
600-699	TBD	See UPnP Device Architecture section on Control.
702	Invalid InstanceID	The specified *InstanceID* is invalid.

2.4.25 *GetVerticalKeystone()*

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

Note: this functionality is also offered by the *VerticalKeystone* transform, see Appendix A.16.16.

2.4.25.1 Arguments

Table 2-71: 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 2-72: 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 OPTIONAL action sets the *VerticalKeystone* state variable of the specified instance of this service to the specified value.

Note: this functionality is also offered by the *VerticalKeystone* transform, see Appendix A.16.16.

2.4.26.1 Arguments

Table 2-73: 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 2-74: 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 OPTIONAL action retrieves the current value of the *Mute* setting of the channel for the specified instance of this service.

Note: this functionality is also offered by the *Mute* transform, see Appendix A.16.3.

2.4.27.1 Arguments

Table 2-75: 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 2-76: 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 OPTIONAL action sets the *Mute* state variable of the specified instance of this service to the specified value.

Note: this functionality is also offered by the *Mute* transform, see Appendix A.16.3.

2.4.28.1 Arguments

Table 2-77: 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 2-78: 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 OPTIONAL 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.

Note: this functionality is also offered by the *Volume* transform, see Appendix A.16.1.

2.4.29.1 Arguments

Table 2-79: 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 2-80: 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 OPTIONAL 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.

Note: this functionality is also offered by the *Volume* transform, see Appendix A.16.1.

2.4.30.1 Arguments

Table 2-81: 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 2-82: 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.

errorCode	errorDescription	Description
702	Invalid InstanceID	The specified *InstanceID* is invalid.
703	Invalid Channel	The specified *Channel* is invalid.

2.4.31 *GetVolumeDB()*

This OPTIONAL 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.

Note: this functionality is also offered by the *VolumeDB* transform, see Appendix A.16.2.

2.4.31.1 Arguments

Table 2-83: 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 2-84: 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 OPTIONAL 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.

Note: this functionality is also offered by the *VolumeDB* transform, see Appendix A.16.2.

2.4.32.1 Arguments

Table 2-85: 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 2-86: 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 CONDITIONALLY REQUIRED action MUST be implemented if the *SetVolumeDB()* action is implemented. The 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 2-87: 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 2-88: 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 OPTIONAL action retrieves the current value of the *Loudness* setting of the channel for the specified instance of this service.

Note: this functionality is also offered by the *Loudness* transform, see Appendix A.16.4.

2.4.34.1 Arguments

Table 2-89: 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 2-90: 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 OPTIONAL action sets the specified value of the *Loudness* state variable of the channel for the specified instance of this service to the specified value.

Note: this functionality is also offered by the *Loudness* transform, see Appendix A.16.4.

2.4.35.1 Arguments

Table 2-91: 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 2-92: 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 OPTIONAL 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*, *AllowedTransformSettings*, *TransformSettings* 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 2-93: 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 2-94: 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 OPTIONAL 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*, *PresetNameList*, *AllowedTransformSettings* *AllowedDefaultTransformSettings* and *TransformSettings*) the action MUST return the appropriate error code.

2.4.37.1 Arguments

Table 2-95: 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 2-96: 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 *GetAllowedTransforms()*

This CONDITIONALLY REQUIRED action MUST be implemented if the *GetTransforms()*, *SetTransforms()* or *GetAllAvailableTransforms()* actions are implemented, that is, these four actions MUST be implemented as a combination. The action returns a list of the currently allowed transforms and their allowed arguments. The *CurrentAllowedTransformSettings* OUT argument contains an XML document with the list of allowed transforms and allowed values for the corresponding transforms. The content of the *CurrentAllowedTransformSettings* argument is dependent on the *InstanceID* and the type of content associated with this *InstanceID*. All supported transforms that can be set for the type of content associated with the *InstanceID* MUST be returned.

2.4.38.1 Arguments

Table 2-97: Arguments for *GetAllowedTransforms()*

Argument	Direction	relatedStateVariable
InstanceID	*IN*	*A_ARG_TYPE_InstanceID*
CurrentAllowedTransformSettings	*OUT*	*AllowedTransformSettings*

2.4.38.2 Dependency on State

None.

2.4.38.3 Effect on State

None.

2.4.38.4 Errors

Table 2-98: Error Codes for *GetAllowedTransforms()*

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.

2.4.39 *GetTransforms()*

This CONDITIONALLY REQUIRED action MUST be implemented if the *GetAllowedTransforms()*, *SetTransforms()* or *GetAllAvailableTransforms()* actions are implemented, that is, these four actions MUST be implemented as a combination. The action returns a list of transforms and their corresponding current transform values. The *CurrentTransformValues* argument returns a list of all applicable transforms values for the media associated with the indicated instance.

2.4.39.1 Arguments

Table 2-99: Arguments for *GetTransforms()*

Argument	Direction	relatedStateVariable
InstanceID	*IN*	*A_ARG_TYPE_InstanceID*
CurrentTransformValues	*OUT*	*TransformSettings*

2.4.39.2 Dependency on State

None.

2.4.39.3 Effect on State

None.

2.4.39.4 Errors

Table 2-100: Error Codes for *GetTransforms()*

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.

2.4.40 *SetTransforms()*

This CONDITIONALLY REQUIRED action MUST be implemented if the *GetAllowedTransforms()*, *GetTransforms()* or *GetAllAvailableTransforms()* actions are implemented, that is, these four actions MUST be implemented as a combination. The action applies the desired values of the specified transforms to the indicated instance. The applicable transforms and their allowed values can be obtained by invoking the *GetAllowedTransforms()* action with the same *InstanceID*. The media renderer MUST process the transforms in same the order as the control point provides for the `transform` elements in the *TransformSettings* XML document.

When a set of transforms is being applied in one action, the action will succeed if one or more transforms succeeds. Furthermore, when a transform has been set on a device with an unsupported value, the transform MUST maintain its current value. If one or more transforms have failed, the control point can detect this by examining the event generated by the invocation of the *SetTransforms()* action.

Note: The current values of the transforms remain in effect when reusing the same *InstanceID* for playing the next item.

Note: When the `shared` flag on the transform is set to "*1*" then the transform MUST be set with *InstanceID* = 0.

2.4.40.1 Arguments

Table 2-101: Arguments for *SetTransforms()*

Argument	Direction	relatedStateVariable
InstanceID	*IN*	*A_ARG_TYPE_InstanceID*
DesiredTransformValues	*IN*	*TransformSettings*

2.4.40.2 Dependency on State

None.

2.4.40.3 Effect on State

None.

2.4.40.4 Errors

Table 2-102: Error Codes for *SetTransforms()*

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.
711	Transforms Not Allowed	The specified transforms are not in the allowed list of transforms
712	Unsupported Values for Transforms	The specified transforms cannot be applied because the input value for each specified transforms is not supported.
713	Internal Error	The specified transform cannot be applied because an internal error occurred in the rendering device.

2.4.41 *GetAllowedDefaultTransforms()*

This CONDITIONALLY REQUIRED action MUST be implemented if the *GetDefaultTransforms()* or *SetDefaultTransforms()* actions are implemented, that is, these three actions MUST be implemented as a combination. Furthermore, if these actions are implemented, then the actions *GetAllowedTransforms()*, *GetTransforms()*, *SetTransforms()* and *GetAllAvailableTransforms()* MUST also be implemented. The action returns a list of transforms which permit modification of their current default settings. The *AllowedDefaultTransformSettings* argument returns an XML document containing a list of transforms and their allowed default values. The selection of transforms which can accept new default values is vendor defined. However, transforms returned by this action MUST be selected from the complete list of transforms implemented by the device.

The *GetAllowedDefaultTransforms()* action MUST return all transforms that implements a default setting. However the allowed values for a particular transform can be designed in such a way that the default value cannot be changed (that is, only one value possible).

2.4.41.1 Arguments

Table 2-103: Arguments for *GetAllowedDefaultTransforms()*

Argument	Direction	relatedStateVariable
AllowedDefaultTransformSettings	*OUT*	*AllowedDefaultTransformSettings*

2.4.41.2 Dependency on State

None.

2.4.41.3 Effect on State

None.

2.4.41.4 Errors

Table 2-104: Error Codes for *GetAllowedDefaultTransforms()*

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

2.4.42 ***GetDefaultTransforms()***

This CONDITIONALLY REQUIRED action MUST be implemented if the *GetAllowedDefaultTransforms()* or *SetDefaultTransforms()* actions are implemented, that is, these three actions MUST be implemented as a combination. Furthermore, if these actions are implemented, then the actions *GetAllowedTransforms()*, *GetTransforms()*, *SetTransforms()* and *GetAllAvailableTransforms()* MUST also be implemented. The action returns a list of transforms which implements a default setting. The *CurrentDefaultTransformSettings* argument returns an XML document containing a list of transforms and their current default values.

2.4.42.1 Arguments

Table 2-105: Arguments for *GetDefaultTransforms()*

Argument	Direction	relatedStateVariable
CurrentDefaultTransformSettings	*OUT*	*DefaultTransformSettings*

2.4.42.2 Dependency on State

None.

2.4.42.3 Effect on State

None.

2.4.42.4 Errors

Table 2-106: Error Codes for *GetDefaultTransforms()*

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

2.4.43 ***SetDefaultTransforms()***

This CONDITIONALLY REQUIRED action MUST be implemented if the *GetAllowedDefaultTransforms()* or *GetDefaultTransforms()* actions are implemented, that is, these three actions MUST be implemented as a combination. Furthermore, if these actions are implemented, then the actions *GetAllowedTransforms()*, *GetTransforms()*, *SetTransforms()* and *GetAllAvailableTransforms()* MUST also be implemented. The action changes the default values for the specified transforms on the MediaRenderer. The *DesiredDefaultTransformSettings* argument contains an XML document with the list of transforms and the desired new default values for the corresponding transforms. The list of transforms and their default allowed values can be obtained by invoking the *GetAllowedDefaultTransforms()* action. Invoking the *SetDefaultTransforms()* action will not change the current transform state of the items currently being rendered by the MediaRenderer. However, control points will get notified that the default value(s) for the transform(s) have been changed.

When the default values of a set of transforms are being modified in one action, the action will succeed if the modification of one or more defaults succeeds. Furthermore, when a default of a transform has been set on a device with an unsupported default value, the transform MUST maintain its current default value. If the modifications of the defaults of one or more transforms have failed, the control point can detect this by examining the content of the *DefaultTransformSettings* stated variable, which is evented by the invocation of the *SetDefaultTransforms()* action.

2.4.43.1 Arguments

Table 2-107: Arguments for *SetDefaultTransforms()*

Argument	Direction	relatedStateVariable
DesiredDefaultTransformSettings	*IN*	*DefaultTransformSettings*

2.4.43.2 Dependency on State

None.

2.4.43.3 Effect on State

None.

2.4.43.4 Errors

Table 2-108: Error Codes for *SetDefaultTransforms()*

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.
711	Transforms Not Allowed	The specified transforms are not in the allowed list of transforms
712	Unsupported values for Transforms	The specified transforms cannot be applied because the input value for each specified transforms is not supported.
713	Internal Error	The specified transform cannot be applied because an internal error occurred in the rendering device.

2.4.44 *GetAllAvailableTransforms()*

This CONDITIONALLY REQUIRED action MUST be implemented if the *GetAllowedTransforms()*, *GetTransforms()* or *SetTransforms()* actions are implemented, that is, these four actions MUST be implemented as a combination. The action returns a list of all transforms and their corresponding allowed argument values supported by this implementation. The *AllAllowedTransformSettings* argument contains an XML document with the list of all available transforms and all allowed values for the corresponding transforms. Note, that while the *AllAllowedTransformSettings* argument returns the full set of transforms supported by the RenderingControl service implementation, some transforms may only be applied to certain media types. In addition, the allowed value list of some transforms may change once a media object is bound to a rendering control instance.

2.4.44.1 Arguments

Table 2-109: Arguments for *GetAllAvailableTransforms()*

Argument	Direction	relatedStateVariable
AllAllowedTransformSettings	*OUT*	*AllowedTransformSettings*

2.4.44.2 Dependency on State

None.

2.4.44.3 Effect on State

None.

2.4.44.4 Errors

Table 2-110: Error Codes for *GetAllAvailableTransforms()*

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

2.4.45 Relationships Between Actions

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

2.4.46 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 2-111: 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.

errorCode	errorDescription	Description
711	Transforms Not Allowed	The specified transforms are not in the allowed list of transforms
712	Unsupported Values for Transforms	The specified transforms cannot be applied because the input value for each specified transforms is not supported.
713	Internal Error	The specified transform cannot be applied because an internal error occurred in the rendering device.

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.

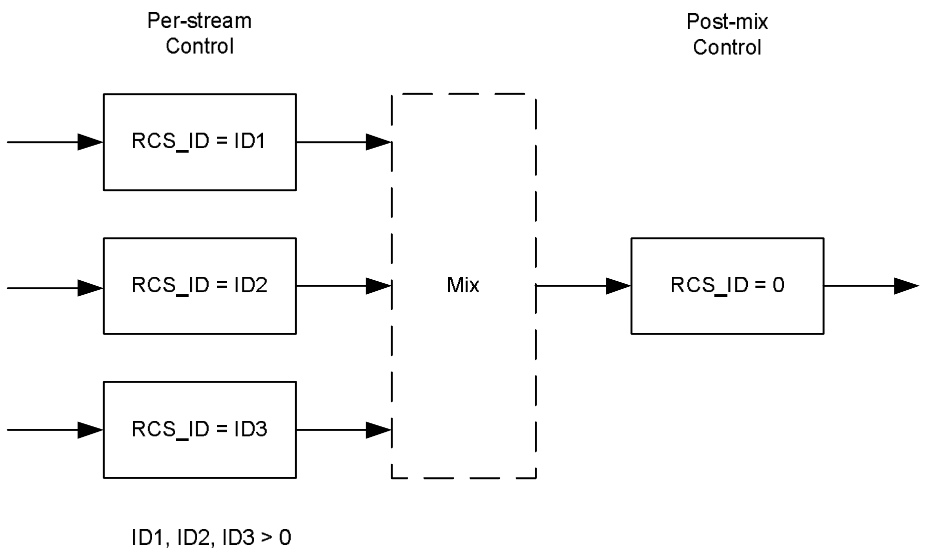

Figure 4: Virtual Instances of RCS

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 2.2.23, “*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.

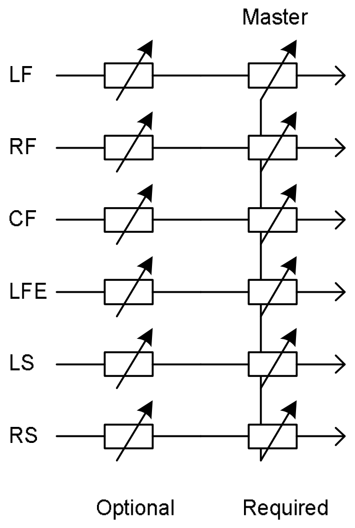

Figure 5: 6-channel Volume Control

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 2.2.31.2, "*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).

2.5.5 Transforms

The RenderingControl service transform action set may be used to perform a number of tasks, for example:

- Change the way a rendering device processes an item. For example, change the rotation angle of an image or adjust the equalization applied to an audio item.
- Change the output configuration of a renderer. For example, enable/disable video outputs, or adjust the current speaker configuration.
- Perform simple stream selection tasks. For example, select an appropriate language track for an audio/video item or select an alternative camera angle from a multiplexed video stream.

A control point can find out which transforms are allowed and which values each transform will accept by invoking the *GetAllowedTransforms()* action. Since the list of available transforms as well as their allowed values can be content dependent, the *GetAllowedTransforms()* action is typically invoked after a media item has been selected to be rendered. The control point can modify the settings of the current transforms by invoking the *SetTransforms()* action on a specific rendering control instance. A control point can check the current status of the transforms by invoking the *GetTransforms()* action.

The control point can use the `<friendlyName>` element of the transform description to describe the transform to an end user. When a transform's friendly name is not supplied by the device implementation, the control point can use the `@name` attribute of the `<transform>` element to describe the transform to an end user.

Each transform can have a default setting, which is determined by the device implementation. A control point can find out which transforms allow their default values to be changed. This can be done by invoking the *GetAllowedDefaultTransforms()* action. The actual setting of a new default value of a transform can be done by invoking the *SetDefaultTransforms()* action.

A control point may monitor a MediaRenderer device's *LastChange* state variable to track transform related state changes. When a new virtual rendering instance is created by the ConnectionManager service *PrepareForConnection()* action, an event due to the RenderingControl service *LastChange* state variable being modified will be generated. The *LastChange* state variable is expected to contain updates to the *AllowedTransformSettings* and *TransformSettings* state variables for the newly created virtual rendering instance. These state variables will list transforms which may be set prior to associating a media item with the virtual renderer instance.

When a media item is selected, an AVTransport service *LastChange* event will be generated reflecting updates the *AVTransportURI* state variable (and possibly the *CurrentTrackURI* state variable). In addition, a (likely unsynchronized) RenderingControl service event will be generated for updates to the *AllowedTransformSettings* and *TransformSettings* state variables through the *LastChange* state variable. As a control point sets new transform settings values using the *SetTransforms()* action, additional RenderingControl service *LastChange* events will be generated reflecting the current value of the *TransformSettings* state variable.

2.5.5.1 Retrieving Transforms

The control point needs to retrieve all supported transforms on a specific MediaRenderer. It does this via the *GetAllAvailableTransforms()* action:

```
Request:
GetAllAvailableTransforms()


Response:
GetAllAvailableTransforms("
   <?xml version="1.0" encoding="UTF-8"?>
   <TransformList
         xmlns="urn:schemas-upnp-org:av:AllowedTransformSettings"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation=
            "urn:schemas-upnp-org:av:AllowedTransformSettings
          http://www.upnp.org/schemas/av/AllowedTransformSettings.xsd">
      <transform name="ClosedCaptioning" shared="0">
         <allowedValueList inactiveValue="en">
            <allowedValue>None</allowedValue>
            <allowedValue>en</allowedValue>
            <allowedValue>fr</allowedValue>
            <allowedValue>nl</allowedValue>
            <allowedValue>und</allowedValue>
         </allowedValueList>
      </transform>
      <transform name="AudioTrackSelection" shared="0">
         <allowedValueList inactiveValue="None">
            <allowedValue>None</allowedValue>
            <allowedValue>en</allowedValue>
            <allowedValue>nl</allowedValue>
            <allowedValue>zxx</allowedValue>
         </allowedValueList>
      </transform>
      <transform name="Rotation" shared="0">
         <allowedValueRange inactiveValue="0" unit="deg">
            <minimum>0</minimum>
            <maximum>270</maximum>
            <step>90</step>
         </allowedValueRange>
      </transform>
      <transform name="Brightness" shared="1">
         <allowedValueRange inactiveValue="50">
            <minimum>0</minimum>
            <maximum>100</maximum>
            <step>1</step>
         </allowedValueRange>
      </transform>
      <transform name="Zoom" shared="0">
         <allowedValueRange inactiveValue="100" unit="pct">
            <minimum>0</minimum>
            <maximum>400</maximum>
            <step>10</step>
         </allowedValueRange>
      </transform>
      <transform name="Volume_Master" shared="1">
         <allowedValueRange inactiveValue="0">
            <minimum>0</minimum>
            <maximum>100</maximum>
         </allowedValueRange>
      </transform>
      <transform name="X_Philips.com_AmbiLight" shared="1">
         <friendlyName>Ambilight responsiveness</friendlyName>
         <allowedValueList inactiveValue="Normal">
            <allowedValue>Slow</allowedValue>
            <allowedValue>Normal</allowedValue>
            <allowedValue>Fast</allowedValue>
         </allowedValueList>
      </transform>
   </TransformList>
")
```

2.5.5.2 Get Allowed Transforms from an instance

The control point needs to retrieve the transforms on a specific MediaRenderer when playing an image on *instanceID* = 1. It does this via the *GetAllowedTransforms()* action:

```
Request:
GetAllowedTransforms(1)

Response:
GetAllowedTransforms("
   <?xml version="1.0" encoding="UTF-8"?>
   <TransformList
         xmlns="urn:schemas-upnp-org:av:AllowedTransformSettings"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation=
            "urn:schemas-upnp-org:av:AllowedTransformSettings
          http://www.upnp.org/schemas/av/AllowedTransformSettings.xsd">
      <transform name="Rotation" shared="0">
         <allowedValueRange inactiveValue="0" unit="deg">
            <minimum>0</minimum>
            <maximum>270</maximum>
            <step>90</step>
         </allowedValueRange>
      </transform>
      <transform name="Zoom" shared="0">
         <allowedValueRange inactiveValue="100" unit="pct">
```

```
            <minimum>0</minimum>
            <maximum>400</maximum>
         </allowedValueRange>
      </transform>
      <transform name="HorizontalPan" shared="0">
         <allowedValueRange inactiveValue="0" unit="px" scale="LINEAR">
            <minimum>-500</minimum>
            <maximum>500</maximum>
         </allowedValueRange>
      </transform>
      <transform name="VerticalPan" shared="0">
         <allowedValueRange inactiveValue="0" unit="px" scale="LINEAR">
            <minimum>-500</minimum>
            <maximum>500</maximum>
         </allowedValueRange>
      </transform>
   </TransformList>
")
```

The control point needs to retrieve the transforms on a specific MediaRenderer when playing a video on *instanceID* = 2. It does this via the *GetAllowedTransforms()* action:

Request:
```
GetAllowedTransforms(2)
```

Response:
```
GetAllowedTransforms("
   <?xml version="1.0" encoding="UTF-8"?>
   <TransformList
      xmlns="urn:schemas-upnp-org:av:AllowedTransformSettings"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation=
         "urn:schemas-upnp-org:av:AllowedTransformSettings
          http://www.upnp.org/schemas/av/AllowedTransformSettings.xsd">
      <transform name="Volume_Master" shared="1">
         <allowedValueRange inactiveValue="0">
            <minimum>0</minimum>
            <maximum>100</maximum>
            <step>1</step>
         </allowedValueRange>
      </transform>
      <transform name="HorizontalPan" shared="0">
         <allowedValueRange
            inactiveValue="0"
            unit="px"
            scale="LINEAR"
            info="Horizontal pan (pixels)">
            <minimum>-500</minimum>
            <maximum>500</maximum>
         </allowedValueRange>
         <allowedValueRange
            inactiveValue="0"
            unit="pct"
            scale="LINEAR"
            info="Horizontal pan (percent)">
            <minimum>-25</minimum>
            <maximum>25</maximum>
         </allowedValueRange>
      </transform>
   </TransformList>
")
```

The result indicates that for *InstanceID* = 2, 2 transforms can be set, *Volume_Master* and *HorizontalPan*. There are two sets of allowed value ranges for transform *HorizontalPan*, each with a different unit. When setting this transform, the control point has a choice between specifying the value as a number of pixels or as a percentage.

2.5.5.3 Setting Transforms

The control point desires to set an image *Rotation* transform to 90 degrees on *InstanceID* = 3. The control point does this via the *SetTransforms()* action:

```
Request:
SetTransforms(3, "
   <?xml version="1.0" encoding="UTF-8"?>
   <TransformSettings
      xmlns="urn:schemas-upnp-org:av:TransformSettings"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation=
         "urn:schemas-upnp-org:av:TransformSettings
           http://www.upnp.org/schemas/av/TransformSettings.xsd">
      <transform name="Rotation">
         <value>90</value>
      </transform>
   </TransformSettings>
")

Response:
SetTransforms()
```

For *InstanceID* = 2 (see Section 2.5.5.2), where the *HorizontalPan* transform provides two different units, the control point sets the transform to a value as a number of pixels. This is done as follows:

```
Request:
SetTransforms(2, "
   <?xml version="1.0" encoding="UTF-8"?>
   <TransformSettings
      xmlns="urn:schemas-upnp-org:av:TransformSettings"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation=
         "urn:schemas-upnp-org:av:TransformSettings
          http://www.upnp.org/schemas/av/TransformSettings.xsd">
      <transform name="HorizontalPan">
         <value index="0">100</value>
      </transform>
   </TransformSettings>
")

Response:
SetTransforms()
```

2.5.5.4 Retrieving Current values of the Transforms

The control point desires to retrieving the values of the currently applied transforms on *InstanceID* = 1. It does that by the *GetTransforms()* action:

Request:
```
GetTransforms(1)
```

Response:
```
GetTransforms("
   <?xml version="1.0" encoding="UTF-8"?>
   <TransformSettings
      xmlns="urn:schemas-upnp-org:av:TransformSettings"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation=
         "urn:schemas-upnp-org:av:TransformSettings
           http://www.upnp.org/schemas/av/TransformSettings.xsd">
      <transform name="ClosedCaptioning">
         <value>fr</value>
      </transform>
      <transform name="AudioTrackSelection">
         <value>en</value>
      </transform>
   </TransformSettings>
")
```

2.5.5.5 Querying and setting default values for a Transform

The control point desires to set the default value of the image *Rotation* transform to 90 degrees. Before setting the default value, the control point first checks whether this implementation allows the *Rotation* transform to change its default value and that the value of 90 degrees can be used as a default value. The control point does this via the following requests of *GetAllowedDefaultTransforms()* and *SetDefaultTransforms()* actions:

Request:
```
GetAllowedDefaultTransforms()
```

Response:
```
GetAllowedDefaultTransforms("
   <?xml version="1.0" encoding="UTF-8"?>
   <TransformList
      xmlns="urn:schemas-upnp-org:av:AllowedTransformSettings"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation=
         "urn:schemas-upnp-org:av:AllowedTransformSettings
          http://www.upnp.org/schemas/av/AllowedTransformSettings.xsd">
      <transform name="ClosedCaptioning">
         <allowedValueList inactiveValue="None">
            <allowedValue>None</allowedValue>
            <allowedValue>en</allowedValue>
            <allowedValue>nl</allowedValue>
            <allowedValue>fr</allowedValue>
            <allowedValue>en-US</allowedValue>
            <allowedValue>und</allowedValue>
         </allowedValueList>
      </transform>
      <transform name="Rotation" shared="0">
         <allowedValueRange inactiveValue="0" unit="deg">
```

```
            <minimum>0</minimum>
            <maximum>270</maximum>
            <step>90</step>
         </allowedValueRange>
      </transform>
   </TransformList>
")
```

Request:
```
SetDefaultTransforms("
   <?xml version="1.0" encoding="UTF-8"?>
   <TransformSettings
      xmlns="urn:schemas-upnp-org:av:TransformSettings"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation=
         "urn:schemas-upnp-org:av:TransformSettings
          http://www.upnp.org/schemas/av/TransformSettings.xsd">
      <transform name="Rotation">
         <value>90</value>
      </transform>
   </TransformSettings>
")
```

Response:
```
SetDefaultTransforms()
```

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>
      <action>
         <name>GetAllowedTransforms</name>
         <argumentList>
            <argument>
               <name>InstanceID</name>
               <direction>in</direction>
               <relatedStateVariable>
                  A_ARG_TYPE_InstanceID
               </relatedStateVariable>
            </argument>
            <argument>
               <name>CurrentAllowedTransformSettings</name>
               <direction>out</direction>
               <relatedStateVariable>
                  AllowedTransformSettings
               </relatedStateVariable>
            </argument>
         </argumentList>
      </action>
      <action>
         <name>SetTransforms</name>
         <argumentList>
            <argument>
               <name>InstanceID</name>
               <direction>in</direction>
               <relatedStateVariable>
                  A_ARG_TYPE_InstanceID
               </relatedStateVariable>
            </argument>
            <argument>
               <name>DesiredTransformValues</name>
               <direction>in</direction>
               <relatedStateVariable>
                  TransformSettings
               </relatedStateVariable>
            </argument>
         </argumentList>
```

```
</action>
<action>
   <name>GetTransforms</name>
   <argumentList>
      <argument>
         <name>InstanceID</name>
         <direction>in</direction>
         <relatedStateVariable>
            A_ARG_TYPE_InstanceID
         </relatedStateVariable>
      </argument>
      <argument>
         <name>CurrentTransformValues</name>
         <direction>out</direction>
         <relatedStateVariable>
            TransformSettings
         </relatedStateVariable>
      </argument>
   </argumentList>
</action>
<action>
   <name>GetAllAvailableTransforms</name>
   <argumentList>
      <argument>
         <name>AllAllowedTransformSettings</name>
         <direction>out</direction>
         <relatedStateVariable>
            AllowedTransformSettings
         </relatedStateVariable>
      </argument>
   </argumentList>
</action>
<action>
   <name>GetAllowedDefaultTransforms</name>
   <argumentList>
      <argument>
         <name>AllowedDefaultTransformSettings</name>
         <direction>out</direction>
         <relatedStateVariable>
            AllowedDefaultTransformSettings
         </relatedStateVariable>
      </argument>
   </argumentList>
</action>
<action>
   <name>GetDefaultTransforms</name>
   <argumentList>
      <argument>
         <name>CurrentDefaultTransformSettings</name>
         <direction>out</direction>
         <relatedStateVariable>
            DefaultTransformSettings
         </relatedStateVariable>
      </argument>
   </argumentList>
</action>
<action>
```

```
         <name>SetDefaultTransforms</name>
         <argumentList>
            <argument>
               <name>DesiredDefaultTransformSettings</name>
               <direction>in</direction>
               <relatedStateVariable>
                  DefaultTransformSettings
               </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>BC</allowedValue>
            <allowedValue>BL</allowedValue>
            <allowedValue>BR</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>
      <stateVariable sendEvents="no">
         <name>AllowedTransformSettings</name>
         <dataType>string</dataType>
      </stateVariable>
      <stateVariable sendEvents="no">
         <name>TransformSettings</name>
         <dataType>string</dataType>
      </stateVariable>
      <stateVariable sendEvents="yes">
         <name>AllowedDefaultTransformSettings</name>
         <dataType>string</dataType>
      </stateVariable>
      <stateVariable sendEvents="yes">
         <name>DefaultTransformSettings</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.

Appendix A. Pre-defined Transforms (normative)

This appendix lists a set of normative transform definitions. Implementations MAY choose which of these transforms to support, however if they support any of the transforms listed here, then the implementation MUST adhere to the definition as given in this appendix.

The complete list of transforms is summarized in Table 4-1. For a detailed description see further subsections.

Table 4-1: Pre-defined Transforms

Name	Description	Typical Content Type
Rotation	Rotates an image on the screen	Image, Video
RedEye	Removes red eye in images	Image
Zoom	Zoom factor on a screen	Image, Video
HorizontalPan	Horizontal pan factor on the screen	Image, Video
VerticalPan	Vertical pan factor on the screen	Image, Video
Equalization	Equalization of sound as total effect	Audio, Video
BandEq_[XX]_[YY]	Equalization per band. From range xx to yy in Hz.	Audio, Video
SpeakerConfiguration	Audio output modes, how the content will be rendered on the speakers	Audio, Video
OutputSelection_[Name]	Audio output selection.	Audio, Video
AudioTrackSelection	Select an audio track language from a range of available audio tracks	Audio, Video
ClosedCaptioning	Selects closed captioning from a video stream	Video
Subtitle	Selects a subtitle from a video stream	Video
CameraAngle	Selects a camera angle from a video stream	Video
PiP	Sets picture in picture mode	Image, Video
ComponentInfoSelection	Selects a user experience	Audio, Video
Volume_[Channel]	Sets the volume	Audio, Video
VolumeDB_[Channel]	Sets the volume in decibels	Audio, Video
Mute_[Channel]	Sets the mute on or off	Audio, Video
Loudness_[Channel]	Sets the loudness on or off	Audio, Video
Brightness	Sets the brightness	Image, Video
Sharpness	Sets the sharpness	Image, Video
Contrast	Sets the contrast	Image, Video
RedVideoGain	Sets the red gain control level	Image, Video

Name	Description	Typical Content Type
GreenVideoGain	Sets the green gain control level	Image, Video
BlueVideoGain	Sets the blue gain control level	Image, Video
RedVideoBlacklevel	Sets the minimum intensity of red	Image, Video
GreenVideoBlacklevel	Sets the minimum intensity of green	Image, Video
BlueVideoBlacklevel	Sets the minimum intensity of blue	Image, Video
ColorTemperature	Sets the color quality of white	Image, Video
HorizontalKeystone	Sets the compensation for horizontal distortion	Image, Video
VerticalKeystone	Sets the compensation for vertical distortion	Image, Video
Vendor-defined		

Vendors MAY define their own transforms. Vendor-defined transform names MUST be prefixed with "X_", followed by the name of the vendor, and followed by an underscore. The vendor name SHOULD be the ICANN name. An example for such a transform is "*X_Philips.com_AmbiLight*".

A.1 *Rotation*

This transform indicates the rotation of a displayed image. The numeric rotation angle is RECOMMENDED to be specified in degrees, and is interpreted as a rotation of the image on screen within the display window in the clockwise direction. The *Rotation* transform is always performed from the center of the image.

Table 4-2: Recommended properties for *Rotation*

Property name	Value
@unit	deg
type	Numeric
friendlyName	Rotation
@scale	Linear
@info	Linear rotation in degrees
@inactiveValue	0
Default value	0

Table 4-3: allowedValueRange for *Rotation*

	Value	R/O
minimum	*0*	*R*
maximum	*Vendor-defined, but at most 359*	*R*
step	*Vendor-defined*	*O*

A.2 RedEye

This transform indicates the algorithm to be used for removal of red eyes in a displayed image. The set of algorithms available is vendor defined.

Table 4-4: Recommended properties for *RedEye*

Property name	Value
@unit	N/A
type	string
friendlyName	Red eye correction
@scale	N/A
@info	Red eye correction
@inactiveValue	Off
Default value	Off

Table 4-5: allowedValueList for *RedEye*

Value	R/O	Description
"*Off*"	*R*	No red eye removal algorithm is active.
"*On*"	*O*	The red eye removal algorithm is active
Vendor-defined		

The value "*On*" MUST be used when only one RedEye removal algorithm will be implemented by the vendor. When more than one RedEye removal algorithm is offered the name of the algorithms MUST be provided instead of the value "*On*". The "*Off*" value MUST always be supported.

A.3 Zoom

This transform indicates the zoom factor of a displayed image or video. For the RECOMMENDED unit scale, percentage, a zoom factor of 100 indicates that the zoom algorithm is inactive. If the zoom factor is larger than 100, then the displayed image is shown larger than original. For example, when a zoom factor of 200 is used, the image width and height are twice the length as the original image, resulting in an image with area size of 4 times bigger than the original image. If the zoom factor is smaller than 100, then the displayed image is shown smaller than original. For example when a zoom factor of 50 is used, the image width and height are half of the original image, resulting in an image with area size of 4 times smaller than the original image.

The *Zoom* transform always works from the center of the image. The *Zoom* transform preserves the aspect ratio of the displayed image.

Table 4-6: Recommended properties for *Zoom*

Property name	Value
@unit	pct (percentage)
type	Numeric
friendlyName	Zoom

Property name	Value
@scale	Linear
@info	Zoom factor in percent
@inactiveValue	100
Default value	100

Table 4-7: allowedValueRange for *Zoom*

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

For implementations that only support a limited set of discrete scaling factors, an `<allowedValueList>` element MAY be used to convey percentages for the *Zoom* transform.

A.3.1 Additional units for the *Zoom* Transform

In addition to the percentage scale, an implementation MAY support an additional unit to provide named preset zoom values. The selection of pre-defined zoom values and the corresponding end-user information strings are vendor dependent, for example “FitScreen”, “Auto”, “WideFit”. When a pre-defined value is chosen the percentage unit scale of the transform MUST provide the actual scaling applied. For example, if the pre-defined value “WideFit” is selected, the implementation calculates the percentage value that is needed to achieve the desired scaling.

Table 4-8: Alternative properties for *Zoom*

Property name	Value
@unit	N/A
type	String
@scale	N/A
@info	Pre-defined zoom values
@inactiveValue	None

Table 4-9: allowedValueList for *Zoom* pre-defined values

Value	R/O	Description
“*N/A*“	*R*	A scaling percentage was selected that does not correspond to any pre-defined value.
“*None*”	*R*	No scaling applied (100 percent)
Vendor-defined	*O*	Pre-defined zoom names with vendor defined behavior

A.4 *HorizontalPan*

This transform indicates a deviation of the center of the display window, relative to the center of the image in the horizontal direction. In other words, the viewing window is shifted horizontally. The value 0 means that the center of the image coincides with the center of display window. A positive value means that the right side of the image is more in view in the display window. A negative value means that the left side of the image is more in view in the display window. It is RECOMMENDED that the range specified for this transform has a linear behavior. For example, a setting of 100 shifts the image twice as far as a value of 50. Furthermore the range MUST be symmetrical (for example, **minimum** value MUST be the negative **maximum** value). It is RECOMMENDED that the **step** has a value of 1.

Table 4-10: Recommended properties for *HorizontalPan*

Property name	Value
@unit	px (pixels)
type	Numeric
friendlyName	Horizontal Pan
@scale	Linear
@info	Horizontal Pan (pixels)
@inactiveValue	0
Default value	0

Table 4-11: allowedValueRange for *HorizontalPan*

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

A.4.1 Additional units for the HorizontalPan Transform

Concurrent with the pixel unit, the *HorizontalPan* transform MAY support the following units:

A.4.1.1 pct (percentage)

A percentage value of 0 indicates that the image center corresponds to the viewport center. A negative percentage value means that the left side of the image is more in view in the display window, while a positive percentage value indicates that the right side of the image is more in view. For example, a percentage value of -50 indicates that the contents of the left side of the viewport have been shifted to the center of the viewport. A percentage value of -100 indicates that content at the left edge of the viewport is now positioned at the right edge of the viewport. *HorizontalPan* values exceeding +/- 100 percent are allowed since the image width may exceed the size of the viewport.

Table 4-12: Alternative properties for *HorizontalPan* (percentage unit)

Property name	Value
@unit	pct (percentage)
type	Numeric
@scale	Linear
@info	Horizontal Pan (percent)

Table 4-13: allowedValueRange for *HorizontalPan* (percentage unit)

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

A.4.1.2 mm (distance in millimeters)

A distance value of 0 indicates that the image center corresponds to the viewport center. A negative distance value means that the left side of the image is more in view in the display window, while a positive distance value indicates that the right side of the image is more in view. For example, a value of -75 mm moves the image 7.5 cm to the right. The values for this unit reflect actual displacements on the viewport rather than any scaling information attached to the content. It is RECOMMENDED that the minimum and maximum values are set as indicated in Table 4-16.

Table 4-14: Alternative properties mm for *HorizontalPan*

Property name	value
@unit	mm (millimeters)
type	Numeric
@scale	Linear
@info	Horizontal Pan (mm)

Table 4-15: allowedValueRange for *HorizontalPan (mm)*

	Value	R/O
minimum	*-(image width + viewport size) / 2*	*R*
maximum	*+(image width + viewport size) / 2*	*R*
step	*Vendor-defined*	*O*

A.5 *VerticalPan*

This transform indicates a deviation of the center of the display window relative to the center of the image in the vertical direction. In other words, the viewing window is shifting vertically. The value 0 means that the center of the image coincide with the center of the display window. A positive value means that the upper portion of the image is more in view in the display window. A negative value means that the lower portion of the image is more in view in the display window.

It is RECOMMENDED that the range specified for this transform has a linear behavior. For example, a setting of 100 shifts the image twice as far as a value of 50. Furthermore the range MUST be symmetrical (for example, **minimum** value MUST be the negative **maximum** value). It is RECOMMENDED that the step size has a value of 1.

Table 4-16: Recommended properties for *VerticalPan*

Property name	Value
@unit	px (pixels)
type	Numeric
friendlyName	Vertical Pan

Property name	Value
@scale	Linear
@info	Vertical Pan (pixels)
@inactiveValue	0
Default value	0

Table 4-17: allowedValueRange for *VerticalPan*

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

A.5.1 Additional units for the *VerticalPan* Transform

Concurrent with the pixel unit, the *VerticalPan* transform, MAY support the following units:

A.5.1.1 pct (percentage)

A percentage value of 0 indicates the image center corresponds to the viewport center. A negative percentage value means that the lower portion of the image is more in view in the display window, while a positive percentage value indicates that the upper portion of the image is more in view. For example, a percentage value of -50 indicates that the contents of bottom of the viewport have been shifted to the center of the viewport. A percentage value of 100 indicates that content at the bottom edge of the viewport is now positioned at the top edge of the viewport. *VerticalPan* values exceeding +/- 100 percent are allowed since the image height may exceed the size of the viewport.

Table 4-18: Alternative properties for *VerticalPan* (percentage unit)

Property name	Value
@unit	pct (percentage)
type	Numeric
@scale	Linear
@info	Vertical Pan (percent)

Table 4-19: allowedValueRange for *VerticalPan* (percentage unit)

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

A.5.1.2 mm (distance in millimeters)

A distance value of 0 indicates the image center corresponds to the viewport center. A negative distance value means that the bottom of the image is more in view in the display window, while a positive distance value indicates that the top of the image is more in view. For example a value of -75 mm moves the bottom of the image 7.5 cm up. The values for this unit reflect actual displacements on the viewport rather than any scaling information attached to the content. It is RECOMMENDED that the minimum and maximum values are set as indicated in Table 4-20.

Table 4-20: Alternative properties for *VerticalPan* (mm unit)

Property name	Value
@unit	mm (millimeters)
type	Numeric
@scale	Linear
@info	Vertical Pan (mm)

Table 4-21: allowedValueRange for *VerticalPan* (mm unit)

	Value	R/O
minimum	*-(image height + viewport size) / 2*	*R*
maximum	*+(image height + viewport size) / 2*	*R*
step	*Vendor-defined*	*O*

A.6 *Equalization*

This transform identifies an equalization setting that transforms the sound. The user can choose a specific algorithm to match his preference. The set of equalization values and the corresponding algorithms to change the sound are vendor defined.

Table 4-22: Recommended properties for *Equalization*

Property name	Value
@unit	N/A
type	String
friendlyName	Equalization
@scale	N/A
@info	Equalization by presets
@inactiveValue	Off
Default value	*Vendor-defined*

Table 4-23: allowedValueList for *Equalization*

Value	R/O	Description
"*Off*"	*R*	No sound enhancement is being made
"*Custom*"	*O*	Manual Equalization, by means of the Band equalization. The equalization by presets is turned off.
"*Jazz*"	*O*	Equalizer settings optimized for music classified as Jazz as defined by the vendor
"*Techno*"	*O*	Equalizer settings optimized for music classified

Value	R/O	Description
		as Techno as defined by the vendor
"Rock"	O	Equalizer settings optimized for music classified as Rock as defined by the vendor
"Classical"	O	Equalizer settings optimized for music classified as Classical as defined by the vendor
"R&B"	O	Equalizer settings optimized for music classified as R&B as defined by the vendor
"Country"	O	Equalizer settings optimized for music classified as Country as defined by the vendor
"Speech"	O	Equalizer settings optimized for speech as defined by the vendor
"ConcertHall"	O	Equalizer settings optimized for a Concert Hall sound experience as defined by the vendor
"VirtualSurround"	O	Equalizer settings optimized for a surround sound experience as defined by the vendor
Vendor-defined		

A.7 BandEq_[XX]_[YY]

This transform indicates a band equalization of the frequency band specified by XX and YY, where XX is an integer value specifying the lower bound of the frequency band and YY is an integer value specifying the upper bound of the frequency band (both in Hz). Multiple frequency bands are needed to span the complete frequency range. The number of bands and the total frequency range covered is vendor specific.

Example of a series of transforms with different values for XX and YY, which together represent frequency bands that are needed to cover a frequency range of 0-20000 Hz:

- BandEq_0_50
- BandEq_50_1000
- BandEq_1000_3000
- BandEq_3000_10000
- BandEq_10000_20000

Each transform has an allowed value range to indicate the amplification. If the equalization is modeled using the dB unit, the transform is modeled after the VolumeDB state variable (see section 2.2.17, "VolumeDB").

If the equalization is modeled using a vendor-defined unit, the transform is modeled after the Volume state variable (see section 2.2.16, "Volume").

Table 4-24: Recommended properties for BandEq_[XX]_[YY]

Property name	Value
@unit	dB
type	Numeric
friendlyName	Equalization XX – YY
@scale	Logarithmic

Property name	Value
@info	Equalization XX - YY
@inactiveValue	*Vendor-defined*
Default value	*Vendor-defined*

Table 4-25: allowedValueRange for each *BandEq_[XX]_[YY]*

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

A.7.1 Additional units for the *BandEq_[XX]_[YY]* Transform

Concurrent with the dB unit, the *BandEq_[XX]_[YY]* transform, MAY support the following unit:

A.7.1.1 Equalization Volume (0-100)

The Equalization Volume unit is an implementation dependent mapping to volume settings. It is RECOMMENDED that the value 0 represent no output and the value 100 represents the maximum allowable output for the equipment.

Table 4-26: Alternative properties for *BandEq_[XX]_[YY]* (Equalization volume unit)

Property name	Value
@unit	N/A
type	Numeric
@scale	Linear
@info	Equalization XX - YY (0-100)

Table 4-27: allowedValueRange for *BandEq_[XX]_[YY]* (Equalization volume unit)

	Value	R/O
minimum	*0*	*R*
maximum	*100*	*R*
step	*Vendor-defined*	*O*

A.8 *SpeakerConfiguration*

This transform indicates the different possible audio output modes of a device. These output modes are related to the loudspeaker configuration attached to the rendering device.

Table 4-28: Recommended properties for *SpeakerConfiguration*

Property name	Value
@unit	N/A
type	String

Property name	Value
friendlyName	Speaker configuration
@scale	N/A
@info	Speaker configuration
@inactiveValue	N/A
Default value	*Vendor-defined*

Table 4-29: allowedValueList for *SpeakerConfiguration*

Value	R/O	Description
"*Mono*"	*O*	One channel audio configuration
"*Stereo*"	*O*	2 channel audio configuration, conveying sound through a LF, RF loudspeaker configuration
"*3D-audio*"	*O*	2 channel audio configuration, conveying sound through a LF, RF, CF loudspeaker configuration
"*Quadraphonic*"	*O*	4 channel audio configuration, conveying sound through a LF, RF, LS, RS loudspeaker configuration
"*5.1*"	*O*	6 channel audio configuration, conveying sound through a LF, RF, CF, LS, RS, LFE loudspeaker configuration
"*6.1*"	*O*	7 channel audio configuration, conveying sound through a LF, RF, CF, LS, RS, BC, LFE loudspeaker configuration
"*7.1*"	*O*	8 channel audio configuration, conveying sound through a LF, RF, CF, LS, RS, BL, BR, LFE loudspeaker configuration
Vendor-defined		

A.9 *OutputSelection_[Name]*

This transform indicates the different possible output selections of a device. The output selected by this set of transforms is identified by [Name], where Name identifies the output (for example "Scart", "HDMI").

Table 4-30: Transform names based on *OutputSelection_[Name]*

Transform names	Description
"*OutputSelection_Scart-[N]*"	Scart number N
"*OutputSelection_HDMI-[N]*"	HDMI number N
"*OutputSelection_Analog-[N]*"	Analog number N
"*OutputSelection_RGB-[N]*"	RGB number N
"*OutputSelection_SPDIF-[N]*"	SPDIF number N
Vendor-defined	Transforms for selection of vendor-defined output names, MUST start with "*OutputSelection_*".

Table 4-31: Recommended properties for OutputSelection_[Name]

Property name	Value
@unit	N/A
type	String
friendlyName	OutputSelection – [Name] - [N]
@scale	N/A
@info	OutputSelection – [Name] - [N]
@inactiveValue	*Vendor-defined*
Default Value	*Vendor-defined*

Table 4-32: allowedValueList for OutputSelection_[Name]

Value	R/O	Description
"Off"	R	Output disabled
"On"	R	Output enabled
Vendor-defined		

A.10 AudioTrackSelection

This transform allows selection of an audio language when multiple audio streams can be selected in a multi-component stream. The language definitions are defined in RFC 3066. As example the following values are defined:

- "en" (English)
- "en-US" (American English)
- "fr" (French)
- "nl" (Dutch)
- "zxx" (not known to be a language)
- "und" (undetermined language)

Table 4-33: Recommended properties for AudioTrackSelection

Property name	Value
@unit	N/A
type	String
friendlyName	Audio track selection
@scale	N/A
@info	Audio track language
@inactiveValue	*Vendor-defined*
Default value	*Vendor-defined*

Table 4-34: allowedValueList for *AudioTrackSelection* (language mode)

Value	R/O	Description
[RFC3066] values	*O*	Values defined according to RFC 3066

The renderer MAY also indicate audio tracks by index. This is useful when the language of the track cannot be determined on the renderer.

Table 4-35: Alternative properties for *AudioTrackSelection*

Property name	Value
@unit	N/A
type	String
friendlyName	Audio track selection
@scale	N/A
@info	Audio track number
@inactiveValue	*Vendor-defined*
Default Value	*Vendor-defined*

Table 4-36: allowedValueList for *AudioTrackSelection* (track indication mode)

Value	R/O	Description
"*Track [N]*"	*O*	Track indication when language is not known

The track indication [N] SHOULD be a positive monotone increasing number, preferably starting with the value "1".

A.11 *ClosedCaptioning*

This transform indicates which language of the closed captioning should be used. The language definitions are defined in RFC 3066. As examples the following values are defined:

- "*en*" (English)
- "*en-US*" (American English)
- "*fr*" (French)
- "*nl*" (Dutch)
- "*zxx*" (not known to be a language)
- "*und*" (undetermined language)

Table 4-37: Recommended properties for *ClosedCaptioning*

Property name	Value
@unit	N/A
type	String

Property name	Value
friendlyName	Closed captioning language selection
@scale	N/A
@info	Closed captioning language selection
@inactiveValue	*Vendor-defined*
Default value	*Vendor-defined*

Table 4-38: allowedValueList for *ClosedCaptioning*

Value	R/O	Description
[RFC3066] values	*O*	Values defined according RFC3066

The renderer MAY also indicate closed captioning tracks by index. This is useful when the language of the close captioning cannot be determined on the renderer.

Table 4-39: Alternative properties for *ClosedCaptioning*

Property name	Value
@unit	N/A
type	String
friendlyName	Closed captioning selection
@scale	N/A
@info	Closed captioning number
@inactiveValue	*Vendor-defined*
Default value	*Vendor-defined*

Table 4-40: allowedValueList for *ClosedCaptioning* (indication mode)

Value	R/O	Description
"*CC_[N]*"	*O*	Closed captioning indication when language is not known

The track indication [N] SHOULD be a positive monotone increasing number, preferably starting with the value "1".

A.12 *Subtitle*

This transform indicates which language of the subtitle should be used. The language definitions are defined in RFC 3066. As examples the following values are defined:

- "*en*" (English)
- "*en-US*" (American English)
- "*fr*" (French)
- "*nl*" (Dutch)

- "zxx" (not known to be a language)
- "und" (undetermined language)

Table 4-41: Recommended properties for Subtitle

Property name	Value
@unit	N/A
type	String
friendlyName	Subtitle language selection
@scale	N/A
@info	Subtitle language selection
@inactiveValue	*Vendor-defined*
Default value	*Vendor-defined*

Table 4-42: allowedValueList for Subtitle

Value	R/O	Description
[RFC3066] values	O	Values defined according RFC3066

The renderer MAY also indicate subtitle tracks by index. This is useful when the language of the subtitles cannot be determined on the renderer.

Table 4-43: Alternative properties for Subtitle

Property name	Value
@unit	N/A
type	String
friendlyName	Subtitle selection
@scale	N/A
@info	Subtitle number
@inactiveValue	*Vendor-defined*
Default value	*Vendor-defined*

Table 4-44: allowedValueList for Subtitle (track indication mode)

Value	R/O	Description
"Subtitle_[N]"	O	Subtitle indication when language is not known

The track indication [N] SHOULD be a positive monotone increasing number, preferably starting with the value "1".

A.13 CameraAngle

This transform indicates which camera angle of a multi-video stream should be used.

Table 4-45: Recommended properties for *CameraAngle*

Property name	Value
@unit	N/A
type	String
friendlyName	Camera angle selection
@scale	N/A
@info	Camera angle selection
@inactiveValue	*Vendor-defined*
Default value	*Vendor-defined*

Table 4-46: allowedValueList for *CameraAngle*

Value	R/O	Description
"*None*"	*O*	No camera angle selected
"*CameraAngle_[N]*"	*O*	Camera angle N selected
Vendor defined		

The camera angle indication [N] SHOULD be a positive monotone increasing number, preferably starting with the value "1".

A.14 *PiP*

This transform indicates whether the content rendered on this instance should be shown in a picture in picture window.

Note that the implementation is free to choose more than one PiP window.

Table 4-47: Recommended properties for *PiP*

Property name	Value
@unit	N/A
Type	String
friendlyName	PiP
@scale	N/A
@info	Picture in Picture
@inactiveValue	*Off*
Default value	*Vendor-defined*

Table 4-48: allowedValueList for *PiP*

Value	R/O	Description
"*Off*"	*R*	The PiP widget is "off"
"*On*"	*O*	The PiP widget is displayed

Value	R/O	Description
Vendor-defined		

A.15 ***ComponentInfoSelection***

This transform indicates which content experience should be selected by the media renderer. Multi-streaming content experiences are conveyed by the metadata of an item. A content item can contain more than one experience, which is indicated by having more than one *upnp:componentInfo* elements inside a *upnp:resExt* element of an item. Selecting a *componentInfo* is done by specifying a number, which corresponds to the order in which the *componentInfo* instances appear in the metadata, where the first occurrence has number "1". Information about multi-streaming can be found in the ContentDirectory service specification [CDS].

When a specific *componentInfo* is selected, the *ClosedCaptioning*, *Subtitle* and *AudioTrackSelection* transforms MUST only list the transforms that will make the selection possible of the components inside the selected *componentInfo*.

Table 4-49: Recommended properties for *ComponentInfoSelection*

Property name	Value
@unit	N/A
type	Numeric
friendlyName	Experience selection
@scale	N/A
@info	Experience selection
@inactiveValue	N/A
Default value	1

Table 4-50: allowedValueList for *ComponentInfoSelection*

Value	R/O	Description
Number	*O*	*componentInfo* "number" is selected for playback
Vendor-defined		

A.16 Legacy compatible transforms

This section lists the set of transforms which have the same behavior as defined for certain actions in this specification (for example *GetVolume()*/*SetVolume()*). In order to ensure compatibility with older control points, if a certain transform is supported by an implementation, then the corresponding legacy actions MUST also be supported.

The value and value ranges of these transform MUST have the same values and ranges when the same functionality in the original actions are implemented. Also the current values inside the transforms and the action-related state variables MUST have the same value.

A.16.1 ***Volume_[Channel]***

This transform indicates the volume setting for playback of an audio or video item, where [Channel] indicates the channel on which the setting is applied. The allowed values for [Channel] are the same as for the *A_ARG_TYPE_Channel* state variable, see Section 2.2.23, "*A_ARG_TYPE_Channel*". Examples of valid transform names are:

- Volume_Master
- Volume_LF

This transform has the same behavior as the actions GetVolume() and SetVolume(). Whereas for the actions the channel is given as an input argument, for the transform the channel is integrated in the transform name itself.

Table 4-51: Recommended properties for Volume_[Channel]

Property name	Value
@unit	N/A
type	ui2
friendlyName	Volume
@scale	*Vendor-defined*
@info	Volume setting for channel [Channel]
@inactiveValue	*Vendor-defined*
Default value	*Vendor-defined*

Table 4-52: allowedValueRange for Volume_[Channel]

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

A.16.2 VolumeDB_[Channel]

This transform indicates the volume setting for playback of an audio or video item in decibel (dB), where [Channel] indicates the channel on which the setting is applied. The allowed values for [Channel] are the same as for the A_ARG_TYPE_Channel state variable, see Section 2.2.23, "A_ARG_TYPE_Channel". Examples of valid transform names are:

- VolumeDB_Master
- VolumeDB_LF

This transform has the same behavior as the actions GetVolumeDB() and SetVolumeDB(). Whereas for the actions the channel is given as an input argument, for the transform the channel is integrated in the transform name itself.

Table 4-53: Recommended properties for VolumeDB_[Channel]

Property name	Value
@unit	1/256 dB (decibel)
type	i2
friendlyName	Volume (dB)
@scale	*Vendor-defined*
@info	Volume setting in decibels for channel [Channel]
@inactiveValue	*Vendor-defined*

Property name	Value
Default value	*Vendor-defined*

Table 4-54: allowedValueRange for *VolumeDB_[Channel]*

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

A.16.3 *Mute_[Channel]*

This transform indicates the mute setting for playback of an audio or video item, where [Channel] indicates the channel on which the setting is applied. The allowed values for [Channel] are the same as for the *A_ARG_TYPE_Channel* state variable, see Section 2.2.23, "*A_ARG_TYPE_Channel*". Examples of valid transform names are:

- *Mute_Master*
- *Mute_LF*

This transform has the same behavior as the actions *GetMute()* and *SetMute()*. Whereas for the actions the channel is given as an input argument, for the transform the channel is integrated in the transform name itself.

Table 4-55: Recommended properties for *Mute_[Channel]*

Property name	Value
@unit	N/A
type	**boolean**
friendlyName	Mute
@scale	N/A
@info	Mute the channel [Channel]
@inactiveValue	"*0*"
Default value	"*0*"

Table 4-56: allowedValueList for *Mute_[Channel]*

Value	R/O	Description
"*0*"	*R*	Mute is off
"*1*"	*R*	Mute is on

A.16.4 *Loudness_[Channel]*

This transform indicates the loudness setting for playback of an audio or video item, where [Channel] indicates the channel on which the setting is applied. The allowed values for [Channel] are the same as for the *A_ARG_TYPE_Channel* state variable, see Section 2.2.23, "*A_ARG_TYPE_Channel*". Examples of valid transform names are:

- *Loudness_Master*
- *Loudness_LF*

This transform has the same behavior as the actions *GetLoudness()* and *SetLoudness()*. Whereas for the actions the channel is given as an input argument, for the transform the channel is integrated in the transform name itself.

Table 4-57: Recommended properties for *Loudness_[Channel]*

Property name	Value
@unit	N/A
type	**boolean**
friendlyName	Loudness
@scale	*Vendor-defined*
@info	Loudness effect of channel [Channel] active
@inactiveValue	"*0*"
Default value	"*0*"

Table 4-58: allowedValueList for *Loudness_[Channel]*

Value	R/O	Description
"*0*"	*R*	Loudness effect is off
"*1*"	*R*	Loudness effect is on

A.16.5 *Brightness*

This transform indicates the brightness setting for playback of an image or video item. This transform has the same behavior as the actions *GetBrightness()* and *SetBrightness()*.

Table 4-59: Recommended properties for *Brightness*

Property name	Value
@unit	N/A
type	**ui2**
friendlyName	Brightness
@scale	*Vendor-defined*
@info	Brightness setting
@inactiveValue	*Vendor-defined*
Default value	*Vendor-defined*

Table 4-60: allowedValueRange for *Brightness*

	Value	R/O
minimum	*0*	*R*
maximum	*Vendor-defined*	*R*

	Value	R/O
step	*1*	*R*

A.16.6 *Sharpness*

This transform indicates the sharpness setting for playback of an image or video item. This transform has the same behavior as the actions *GetSharpness()* and *SetSharpness()*.

Table 4-61: Recommended properties for *Sharpness*

Property name	Value
@unit	N/A
type	**ui2**
friendlyName	Sharpness
@scale	*Vendor-defined*
@info	Sharpness setting
@inactiveValue	*Vendor-defined*
Default value	*Vendor-defined*

Table 4-62: allowedValueRange for *Sharpness*

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

A.16.7 *Contrast*

This transform indicates the contrast setting for playback of an image or video item. This transform has the same behavior as the actions *GetContrast()* and *SetContrast()*.

Table 4-63: Recommended properties for *Contrast*

Property name	Value
@unit	N/A
type	**ui2**
friendlyName	Contrast
@scale	*Vendor-defined*
@info	Contrast setting
@inactiveValue	*Vendor-defined*
Default value	*Vendor-defined*

Table 4-64: allowedValueRange for *Contrast*

	Value	R/O
minimum	*0*	*R*

	Value	R/O
maximum	*Vendor-defined*	*R*
step	*1*	*R*

A.16.8 *RedVideoGain*

This transform indicates the red gain control setting for playback of an image or video item. This transform has the same behavior as the actions *GetRedVideoGain()* and *SetRedVideoGain()*.

Table 4-65: Recommended properties for *RedVideoGain*

Property name	Value
@unit	N/A
type	**ui2**
friendlyName	Red video gain
@scale	*Vendor-defined*
@info	Red gain control
@inactiveValue	*Vendor-defined*
Default value	*Vendor-defined*

Table 4-66: allowedValueRange for *RedVideoGain*

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

A.16.9 *GreenVideoGain*

This transform indicates the green gain control setting for playback of an image or video item. This transform has the same behavior as the actions *GetGreenVideoGain()* and *SetGreenVideoGain()*.

Table 4-67: Recommended properties for *GreenVideoGain*

Property name	Value
@unit	N/A
type	**ui2**
friendlyName	Green video gain
@scale	*Vendor-defined*
@info	Green gain control
@inactiveValue	*Vendor-defined*
Default value	*Vendor-defined*

Table 4-68: allowedValueRange for GreenVideoGain

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

A.16.10 BlueVideoGain

This transform indicates the blue gain control setting for playback of an image or video item. This transform has the same behavior as the actions *GetBlueVideoGain()* and *SetBlueVideoGain()*.

Table 4-69: Recommended properties for BlueVideoGain

Property name	Value
@unit	N/A
type	**ui2**
friendlyName	Blue video gain
@scale	*Vendor-defined*
@info	Blue gain control
@inactiveValue	*Vendor-defined*
Default value	*Vendor-defined*

Table 4-70: allowedValueRange for BlueVideoGain

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

A.16.11 RedVideoBlackLevel

This transform indicates the setting for the minimum output intensity of red for playback of an image or video item. This transform has the same behavior as the actions *GetRedVideoBlackLevel()* and *SetRedVideoBlackLevel()*.

Table 4-71: Recommended properties for RedVideoBlackLevel

Property name	Value
@unit	N/A
type	**ui2**
friendlyName	Red video black level
@scale	*Vendor-defined*
@info	Minimum output intensity of red
@inactiveValue	*Vendor-defined*
Default value	*Vendor-defined*

Table 4-72: allowedValueRange for RedVideoBlackLevel

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

A.16.12 *GreenVideoBlackLevel*

This transform indicates the setting for the minimum output intensity of green for playback of an image or video item. This transform has the same behavior as the actions *GetGreenVideoBlackLevel()* and *SetGreenVideoBlackLevel()*.

Table 4-73: Recommended properties for GreenVideoBlackLevel

Property name	Value
@unit	N/A
type	**ui2**
friendlyName	Green video black level
@scale	*Vendor-defined*
@info	Minimum output intensity of green
@inactiveValue	*Vendor-defined*
Default value	*Vendor-defined*

Table 4-74: allowedValueRange for GreenVideoBlackLevel

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

A.16.13 *BlueVideoBlackLevel*

This transform indicates the setting for the minimum output intensity of blue for playback of an image or video item. This transform has the same behavior as the actions *GetBlueVideoBlackLevel()* and *SetBlueVideoBlackLevel()*.

Table 4-75: Recommended properties for BlueVideoBlackLevel

Property name	Value
@unit	N/A
type	**ui2**
friendlyName	Blue video black level
@scale	*Vendor-defined*
@info	Minimum output intensity of blue
@inactiveValue	*Vendor-defined*

Property name	Value
Default value	*Vendor-defined*

Table 4-76: allowedValueRange for *BlueVideoBlackLevel*

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

A.16.14 *ColorTemperature*

This transform indicates the setting for the color quality of white for playback of an image or video item. This transform has the same behavior as the actions *GetColorTemperature()* and *SetColorTemperature()*.

Table 4-77: Recommended properties for *ColorTemperature*

Property name	Value
@unit	N/A
type	**ui2**
friendlyName	Color temperature
@scale	*Vendor-defined*
@info	Color quality of white
@inactiveValue	*Vendor-defined*
Default value	*Vendor-defined*

Table 4-78: allowedValueRange for *ColorTemperature*

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

A.16.15 *HorizontalKeystone*

This transform indicates the setting for the level of compensation for horizontal distortion for playback of an image or video item. This transform has the same behavior as the actions *GetHorizontalKeystone()* and *SetHorizontalKeystone()*.

Table 4-79: Recommended properties for *HorizontalKeystone*

Property name	Value
@unit	N/A
type	**i2**
friendlyName	Horizontal keystone
@scale	*Vendor-defined*
@info	Level of compensation for horizontal distortion

Property name	Value
@inactiveValue	*Vendor-defined*
Default value	*Vendor-defined*

Table 4-80: allowedValueRange for *HorizontalKeystone*

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

A.16.16 *VerticalKeystone*

This transform indicates the setting for the level of compensation for vertical distortion for playback of an image or video item. This transform has the same behavior as the actions *GetVerticalKeystone()* and *SetVerticalKeystone()*.

Table 4-81: Recommended properties for *VerticalKeystone*

Property name	Value
@unit	N/A
type	**i2**
friendlyName	Vertical keystone
@scale	*Vendor-defined*
@info	Level of compensation for vertical distortion
@inactiveValue	*Vendor-defined*
Default value	*Vendor-defined*

Table 4-82: allowedValueRange for *VerticalKeystone*

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