ISO/IEC 29341-4-10

Edition 2.0 2011-09

INTERNATIONAL STANDARD

Information technology – UPnP device architecture – Part 4-10: Audio Video Device Control Protocol – Level 2 – Audio Video Transport Service

ISO/IEC 29341-4-10:2011(E)

IEC Central Office
3, rue de Varembé
CH-1211 Geneva 20
Switzerland
Email: inmail@iec.ch
Web: www.iec.ch

About the IEC

The International Electrotechnical Commission (IEC) is the leading global organization that prepares and publishes International Standards for all electrical, electronic and related technologies.

About IEC publications

The technical content of IEC publications is kept under constant review by the IEC. Please make sure that you have the latest edition, a corrigenda or an amendment might have been published.

- Catalogue of IEC publications: www.iec.ch/searchpub
The IEC on-line Catalogue enables you to search by a variety of criteria (reference number, text, technical committee,…). It also gives information on projects, withdrawn and replaced publications.

- IEC Just Published: www.iec.ch/online_news/justpub
Stay up to date on all new IEC publications. Just Published details twice a month all new publications released. Available on-line and also by email.

- Electropedia: www.electropedia.org
The world's leading online dictionary of electronic and electrical terms containing more than 20 000 terms and definitions in English and French, with equivalent terms in additional languages. Also known as the International Electrotechnical Vocabulary online.

- Customer Service Centre: www.iec.ch/webstore/custserv
If you wish to give us your feedback on this publication or need further assistance, please visit the Customer Service Centre FAQ or contact us:

Email: csc@iec.ch
Tel.: +41 22 919 02 11
Fax: +41 22 919 03 00

ISO/IEC 29341-4-10

Edition 2.0 2011-09

INTERNATIONAL STANDARD

colour inside

Information technology – UPnP device architecture – Part 4-10: Audio Video Device Control Protocol – Level 2 – Audio Video Transport Service

INTERNATIONAL ELECTROTECHNICAL COMMISSION

PRICE CODE W

ICS 35.200

ISBN 978-2-88912-680-4

CONTENTS

INFORMATION TECHNOLOGY – UPNP DEVICE ARCHITECTURE –

Part 4-10: Audio Video Device Control Protocol – Level 2 – Audio Video Transport Service

FOREWORD

1) ISO (International Organization for Standardization) and IEC (International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards. Their preparation is entrusted to technical committees; any ISO and IEC member body interested in the subject dealt with may participate in this preparatory work. International governmental and non-governmental organizations liaising with ISO and IEC also participate in this preparation.

2) In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

3) The formal decisions or agreements of IEC and ISO on technical matters express, as nearly as possible, an international consensus of opinion on the relevant subjects since each technical committee has representation from all interested IEC and ISO member bodies.

4) IEC, ISO and ISO/IEC publications have the form of recommendations for international use and are accepted by IEC and ISO member bodies in that sense. While all reasonable efforts are made to ensure that the technical content of IEC, ISO and ISO/IEC publications is accurate, IEC or ISO cannot be held responsible for the way in which they are used or for any misinterpretation by any end user.

5) In order to promote international uniformity, IEC and ISO member bodies undertake to apply IEC, ISO and ISO/IEC publications transparently to the maximum extent possible in their national and regional publications. Any divergence between any ISO/IEC publication and the corresponding national or regional publication should be clearly indicated in the latter.

6) ISO and IEC provide no marking procedure to indicate their approval and cannot be rendered responsible for any equipment declared to be in conformity with an ISO/IEC publication.

7) All users should ensure that they have the latest edition of this publication.

8) No liability shall attach to IEC or ISO or its directors, employees, servants or agents including individual experts and members of their technical committees and IEC or ISO member bodies for any personal injury, property damage or other damage of any nature whatsoever, whether direct or indirect, or for costs (including legal fees) and expenses arising out of the publication of, use of, or reliance upon, this ISO/IEC publication or any other IEC, ISO or ISO/IEC publications.

9) Attention is drawn to the normative references cited in this publication. Use of the referenced publications is indispensable for the correct application of this publication.

10) Attention is drawn to the possibility that some of the elements of this International Standard may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

International Standard ISO/IEC 29341-4-10 was prepared by UPnP Forum Steering committee[1], was adopted, under the fast track procedure, by subcommittee 25: Interconnection of information technology equipment, of ISO/IEC joint technical committee 1: Information technology.

This International Standard replaces ISO/IEC 29341-4-10, first edition, published in 2008, and constitutes a technical revision.

The list of all currently available parts of the ISO/IEC 29341 series, under the general title *Information technology – UPnP device architecture*, can be found on the IEC web site.

This International Standard has been approved by vote of the member bodies, and the voting results may be obtained from the address given on the second title page.

---

1 UPnP Forum Steering committee, UPnP Forum, 3855 SW 153rd Drive, Beaverton, Oregon 97006 USA. See also "Introduction".

IMPORTANT – The "colour inside" logo on the cover page of this publication indicates that it contains colours which are considered to be useful for the correct understanding of its contents. Users should therefore print this publication using a colour printer.

1 Overview and Scope

1.1 Introduction

This service definition is compliant with the UPnP Device Architecture version 1.0.

This service type enables control over the transport of audio and video streams. The service type defines a common model for A/V transport control suitable for a generic user interface. It can be used to control a wide variety of disc, tape and solid-state based media devices such as CD players, VCRs and MP3 players. A minimal implementation of this service can be used to control tuners.

The service type is related to the ConnectionManager service type, which describes A/V connection setup procedures, and the ContentDirectory service, which offers meta-information about the resource stored on the media. AVTransport also offers an action to retrieve any meta-data embedded in the resource itself.

This service type does not offer *scheduled* recording.

1.2 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.

1.2.1 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 properties be represented as "*0*" and "*1*".

1.2.2 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 Clause 1.3.1, "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

1.2.3 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].

1.2.3.1 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.

Operator	Semantics
|	**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.

1.3 Derived Data Types

This clause 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.

1.3.1 Comma Separated Value (CSV) Lists

The UPnP AV services use state variables, action arguments and properties that represent lists – or one-dimensional arrays – of values. The UPnP Device Architecture, Version 1.0 [DEVICE], does not provide for either an array type or a list type, so a list type is defined here. Lists MAY either be homogeneous (all values are the same type) or heterogeneous (values of different types are allowed). Lists MAY also consist of repeated occurrences of homogeneous or heterogeneous subsequences, all of which have the same syntax and semantics (same number of values, same value types and in the same order). The data type of a homogeneous list is **string** or xsd:string and denoted by CSV (*x*), where *x* is the type of the individual values. The data type of a heterogeneous list is also **string** or xsd:string and denoted by CSV (*x, y, z*), where *x*, *y* and *z* are the types of the individual values. If the number of values in the heterogeneous list is too large to show each type individually, that variable type is represented as CSV (heterogeneous), and the variable description includes additional information as to the expected sequence of values appearing in the list and their corresponding types. The data type of a repeated subsequence list is **string** or xsd:string and denoted by CSV ({*x*, *y*, *z*}), where *x*, *y* and *z* are the types of the individual values in the subsequence and the subsequence MAY be repeated zero or more times.

- A list is represented as a **string** type (for state variables and action arguments) or xsd:string type (for properties).
- Commas separate values within a list.
- Integer values are represented in CSVs with the same syntax as the integer data type specified in [DEVICE] (that is: optional leading sign, optional leading zeroes, numeric 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 Clause 1.2.2, "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
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.

1.4 Management of XML Namespaces in Standardized DCPs

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

The formal name of a namespace is a URI. The namespace prefix used in an XML document is *not* the name of the namespace. The namespace name is, or should be, globally unique. It has a single definition that is accessible to anyone who uses the namespace. It has the same meaning anywhere that it is used, both inside and outside XML documents. The namespace prefix, however, in formal XML usage, is defined only in an XML document. It must be locally unique to the document. Any valid XML no-colon-name may be used. And, in formal XML usage, no two XML documents are ever required to use the same namespace prefix to refer to the same namespace. The creation and use of the namespace prefix was standardized by the W3C XML Committee in [XML-NMSP] strictly as a convenient local shorthand replacement for the full URI name of a namespace in individual documents.

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

For the same reason that namespace prefixes are convenient in XML documents, it is convenient in specification text to refer to namespaces using a namespace prefix. Therefore, this specification declares a “standard” prefix for all XML namespaces used herein. In addition, this specification expands the scope where these prefixes have meaning, beyond a single XML document, to all of its text, XML examples, and certain string-valued properties. This expansion of scope *does not* supercede XML rules for usage in documents, it only augments and complements them in important contexts that are out-of-scope for the XML specifications. 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 Clause 1.4.3, “Namespace Usage Examples” for its associated schema.

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

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

Table 1-3 — Namespace Definitions

Standard Name-space Prefix	Namespace Name	Namespace Description	Normative Definition Document Reference
AV Working Committee defined namespaces			
av	urn:schemas-upnp-org:av:av	Common data types for use in AV schemas	[AV-XSD]
avs	urn:schemas-upnp-org:av:avs	Common structures for use in AV schemas	[AVS-XSD]
avdt	urn:schemas-upnp-org:av:avdt	Datastructure Template	[AVDT]
avt-event	urn:schemas-upnp-org:metadata-1-0/AVT/	Evented *LastChange* state variable for AVTransport	[AVT]
cds-event	urn:schemas-upnp-org:av:cds-event	Evented *LastChange* state variable for ContentDirectory	[CDS]
didl-lite	urn:schemas-upnp-org:metadata-1-0/DIDL-Lite/	Structure and metadata for ContentDirectory	[CDS]
rcs-event	urn:schemas-upnp-org:metadata-1-0/RCS/	Evented *LastChange* state variable for RenderingControl	[RCS]
srs	urn:schemas-upnp-org:av:srs	Metadata and structure for ScheduledRecording	[SRS]
srs-event	urn:schemas-upnp-org:av:srs-event	Evented *LastChange* state variable for ScheduledRecording	[SRS]
upnp	urn:schemas-upnp-org:metadata-1-0/upnp/	Metadata for ContentDirectory	[CDS]
Externally defined namespaces			
dc	http://purl.org/dc/elements/1.1/	Dublin Core	[DC-TERMS]
xsd	http://www.w3.org/2001/XMLSchema	XML Schema Language 1.0	[XML SCHEMA-1] [XML SCHEMA-2]
xsi	http://www.w3.org/2001/XMLSchema-instance	XML Schema Instance Document schema	Clauses 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 [a] • Form 1, Form 2, Form3	Valid Root Element(s)	Schema Reference
AV Working Committee Defined Namespaces			
av	av-v*n-yyyymmdd*.xsd av-v*n*.xsd av.xsd	*n/a*	[AV-XSD]
avs	avs-v*n-yyyymmdd*.xsd avs-vn.xsd avs.xsd	<Capabilities> <Features> <stateVariableValuePairs>	[AVS-XSD]
avdt	avdt-v*n-yyyymmdd*.xsd avdt-vn.xsd avdt.xsd	<AVDT>	[AVDT]
avt-event	avt-event-v*n-yyyymmdd*.xsd avt-event-vn.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]
didl-lite	didl-lite-v*n-yyyymmdd*.xsd didl-lite-vn.xsd didl-lite.xsd	<DIDL-Lite>	[DIDL-LITE-XSD]
rcs-event	rcs-event-v*n-yyyymmdd*.xsd rcs-event-vn.xsd rcs-event.xsd	<Event>	[RCS-EVENT-XSD]
srs	srs-v*n-yyyymmdd*.xsd srs-vn.xsd srs.xsd	<srs>	[SRS-XSD]
srs-event	srs-event-v*n-yyyymmdd*.xsd srs-event-vn.xsd srs-event.xsd	<StateEvent>	[SRS-EVENT-XSD]
upnp	upnp-v*n-yyyymmdd*.xsd upnp-vn.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]

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

1.4.1 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
ScheduledRecording	srs

1.4.2 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 Clause 1.4.3 “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.

Each time a new namespace version is created, a new XML schema document (.xsd) is created and published so that the new namespace definition is represented in a machine-readable form. Since 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"`.

1.4.3 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>
```

1.5 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.

1.5.1 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.

1.5.2 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.

1.5.3 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_".

1.5.4 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.

1.6 References

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

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

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

[AVDT-XSD] – XML Schema for UPnP AV Datastructure Template:1, UPnP Forum, 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-v1.xsd.

[AV-XSD] – *XML Schema for UPnP AV Common XML Data Types*, UPnP Forum, September 30, 2008. Available at: http://www.upnp.org/schemas/av/av-v2-20080930.xsd. Latest version available at: http://www.upnp.org/schemas/av/av-v2.xsd.

[AVS-XSD] – *XML Schema for UPnP AV Common XML Structures*, UPnP Forum, September 30, 2008. Available at: http://www.upnp.org/schemas/av/avs-v2-20080930.xsd. Latest version available at: http://www.upnp.org/schemas/av/avs-v2.xsd.

[AVT] – *AVTransport:2*, UPnP Forum, September 30, 2008. Available at: http://www.upnp.org/specs/av/UPnP-av-AVTransport-v2-Service-20080930.pdf. Latest version available at: http://www.upnp.org/specs/av/UPnP-av-AVTransport-v2-Service.pdf.

[AVT-EVENT-XSD] – *XML Schema for AVTransport:2 LastChange Eventing*, UPnP Forum, 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-v2.xsd.

[CDS] – *ContentDirectory:3*, UPnP Forum, September 30, 2008. Available at: http://www.upnp.org/specs/av/UPnP-av-ContentDirectory-v3-Service-20080930.pdf. Latest version available at: http://www.upnp.org/specs/av/UPnP-av-ContentDirectory-v3-Service.pdf.

[CDS-EVENT-XSD] – XML Schema for ContentDirectory:3 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-v1.xsd.

[CM] – *ConnectionManager:2*, UPnP Forum, September 30, 2008. Available at: http://www.upnp.org/specs/av/UPnP-av-ConnectionManager-v2-Service-20080930.pdf. Latest version available at: http://www.upnp.org/specs/av/UPnP-av-ConnectionManager-v2-Service.pdf.

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

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

[DEVICE] – *UPnP Device Architecture, version 1.0*, UPnP Forum, July 20, 2006. Available at: http://www.upnp.org/specs/architecture/UPnP-DeviceArchitecture-v1.0-20060720.htm. Latest version available at: http://www.upnp.org/specs/architecture/UPnP-DeviceArchitecture-v1.0.htm.

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

[DIDL-LITE-XSD] – XML Schema for ContentDirectory:3 Structure and Metadata (DIDL-Lite), UPnP Forum, September 30, 2008. Available at: http://www.upnp.org/schemas/av/didl-lite-v2-20080930.xsd. Latest version available at: http://www.upnp.org/schemas/av/didl-lite-v2.xsd.

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

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

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

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

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

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

[MR] – *MediaRenderer:2*, UPnP Forum, September 30, 2008. Available at: http://www.upnp.org/specs/av/UPnP-av-MediaRenderer-v2-Device-20080930.pdf. Latest version available at: http://www.upnp.org/specs/av/UPnP-AV-MediaRenderer-v2-Device.pdf.

[MS] – *MediaServer:3*, UPnP Forum, September 30, 2008. Available at: http://www.upnp.org/specs/av/UPnP-av-MediaServer-v3-Device-20080930.pdf. Latest version available at: http://www.upnp.org/specs/av/UPnP-AV-MediaServer-v3-Device.pdf.

[RCS] – *RenderingControl:2*, UPnP Forum, September 30, 2008. Available at: http://www.upnp.org/specs/av/UPnP-av-RenderingControl-v2-Service-20080930.pdf. Latest version available at: http://www.upnp.org/specs/av/UPnP-av-RenderingControl-v2-Service.pdf.

[RCS-EVENT-XSD] –XML Schema for RenderingControl:2 LastChange Eventing, UPnP Forum, September 30, 2008. Available at: http://www.upnp.org/schemas/av/rcs-event-v1-20080930.xsd. Latest version available at: http://www.upnp.org/schemas/av/rcs-event-v1.xsd.

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

[RFC 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.

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

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

[SRS] – *ScheduledRecording:2*, UPnP Forum, September 30, 2008. Available at: http://www.upnp.org/specs/av/UPnP-av-ScheduledRecording-v2-Service-20080930.pdf. Latest version available at: http://www.upnp.org/specs/av/UPnP-av-ScheduledRecording-v2-Service.pdf.

[SRS-XSD] – XML Schema for ScheduledRecording:2 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-v2.xsd.

[SRS-EVENT-XSD] – XML Schema for ScheduledRecording:2 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-v1.xsd.

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

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

[UPNP-XSD] – *XML Schema for ContentDirectory:3 Metadata*, UPnP Forum, September 30, 2008. Available at: http://www.upnp.org/schemas/av/upnp-v3-20080930.xsd. Latest version available at: http://www.upnp.org/schemas/av/upnp-v3.xsd.

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

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

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

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

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

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

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

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

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

[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 ServiceType

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

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

2.2 State Variables

Table 2-6 — State Variables

Variable Name	R/O [a]	Data Type	Allowed Value	Default Value	Eng. Units
TransportState	*R*	**string**	See Table 2-7		
TransportStatus	*R*	**string**	"*OK*", "*ERROR_OCCURRED*", *vendor-defined*		
CurrentMediaCategory	*R*	**string**	See Table 2-8		
PlaybackStorageMedium	*R*	**string**	See Table 2-9		
RecordStorageMedium	*R*	**string**	See Table 2-9		
PossiblePlaybackStorageMedia	*R*	**string**	CSV [b] (**string**)		
PossibleRecordStorageMedia	*R*	**string**	CSV (**string**)		
CurrentPlayMode	*R*	**string**	See Table 2-10	"*NORMAL*"	
TransportPlaySpeed	*R*	**string**	See Clause 2.2.9	"*1*"	
RecordMediumWriteStatus	*R*	**string**	See Table 2-11		
CurrentRecordQualityMode	*R*	**string**	See Table 2-12		
PossibleRecordQualityModes	*R*	**string**	CSV (**string**)		
NumberOfTracks	*R*	**ui4**	See Table 2-13		
CurrentTrack	*R*	**ui4**	See Table 2-14		
CurrentTrackDuration	*R*	**string**			
CurrentMediaDuration	*R*	**string**			
CurrentTrackMetaData	*R*	**string**			
CurrentTrackURI	*R*	**string**			
AVTransportURI	*R*	**string**			
AVTransportURIMetaData	*R*	**string**			
NextAVTransportURI	*R*	**string**			
NextAVTransportURIMetaData	*R*	**string**			
RelativeTimePosition	*R*	**string**			
AbsoluteTimePosition	*R*	**string**			
RelativeCounterPosition	*R*	**i4**			
AbsoluteCounterPosition	*R*	**ui4**			
CurrentTransportActions	*O*	**string**	CSV (**string**) See Table 2-15		
LastChange	*R*	**string**			
DRMState	*O*	**string**	See Table 2-16	"*UNKNOWN*"	
A_ARG_TYPE_SeekMode	*R*	**string**	See Table 2-17		
A_ARG_TYPE_SeekTarget	*R*	**string**			
A_ARG_TYPE_InstanceID	*R*	**ui4**			
A_ARG_TYPE_DeviceUDN	*O*	**string**			
A_ARG_TYPE_ServiceType	*O*	**string**			
A_ARG_TYPE_ServiceID	*O*	**string**			
A_ARG_TYPE_StateVariableValuePairs	*O*	**string**			
A_ARG_TYPE_StateVariableList	*O*	**string**	CSV (**string**)		

Variable Name	R/O [a]	Data Type	Allowed Value	Default Value	Eng. Units
Non-standard state variables implemented by a UPnP vendor go here	*X*	*TBD*	*TBD*	*TBD*	*TBD*

a *R* = REQUIRED, *O* = OPTIONAL, *X* = Non-standard.

b 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.

Table 2-7 — allowedValueList for *TransportState*

Value	R/O
"*STOPPED*"	*R*
"*PLAYING*"	*R*
"*TRANSITIONING*"	*R*
"*PAUSED_PLAYBACK*"	*R*EQUIRED if *Pause()* action is implemented
"*PAUSED_RECORDING*"	*R*EQUIRED if both *Pause()* and *Record()* actions are implemented
"*RECORDING*"	*R*EQUIRED if *Record()* action is implemented.
"*NO_MEDIA_PRESENT*"	*O*
Vendor-defined	

Table 2-8 — allowedValueList for *CurrentMediaCategory*

Value	R/O
"*NO_MEDIA*"	*R*
"*TRACK_AWARE*"	*R*
"*TRACK_UNAWARE*"	*R*

Table 2-9 — allowedValueList for *PlaybackStorageMedium* and *RecordStorageMedium*

Value	R/O	Description
"*UNKNOWN*"	*O*	Unknown medium
"*DV*"	*O*	Digital Video Tape medium
"*MINI-DV*"	*O*	Mini Digital Video Tape medium
"*VHS*"	*O*	VHS Tape medium
"*W-VHS*"	*O*	W-VHS Tape medium
"*S-VHS*"	*O*	Super VHS Tape medium
"*D-VHS*"	*O*	Digital VHS Tape medium
"*VHSC*"	*O*	Compact VHS medium
"*VIDEO8*"	*O*	8 mm Video Tape medium
"*HI8*"	*O*	High resolution 8 mm Video Tape medium
"*CD-ROM*"	*O*	Compact Disc-Read Only Memory medium
"*CD-DA*"	*O*	Compact Disc-Digital Audio medium
"*CD-R*"	*O*	Compact Disc-Recordable medium
"*CD-RW*"	*O*	Compact Disc-Rewritable medium
"*VIDEO-CD*"	*O*	Video Compact Disc medium
"*SACD*"	*O*	Super Audio Compact Disc medium
"*MD-AUDIO*"	*O*	Mini Disc Audio medium
"*MD-PICTURE*"	*O*	Mini Disc Picture medium
"*DVD-ROM*"	*O*	DVD Read Only medium
"*DVD-VIDEO*"	*O*	DVD Video medium
"*DVD+R*"	*O*	DVD Recordable medium
"*DVD-R*"	*O*	DVD Recordable medium
"*DVD+RW*"	*O*	DVD Rewritable medium
"*DVD-RW*"	*O*	DVD Rewritable medium
"*DVD-RAM*"	*O*	DVD RAM medium
"*DVD-AUDIO*"	*O*	DVD Audio medium
"*DAT*"	*O*	Digital Audio Tape medium
"*LD*"	*O*	Laser Disk medium
"*HDD*"	*O*	Hard Disk Drive medium
"*MICRO-MV*"	*O*	Micro MV Tape medium
"*NETWORK*"	*O*	Network Interface medium
"*NONE*"	*O*	No medium present
"*NOT_IMPLEMENTED*"	*O*	Medium type discovery is not implemented
"*SD*"	*O*	SD (Secure Digital) Memory Card medium
"*PC-CARD*"	*O*	PC Card medium
"*MMC*"	*O*	MultimediaCard medium
"*CF*"	*O*	Compact Flash medium
"*BD*"	*O*	Blu-ray Disc medium
"*MS*"	*O*	Memory Stick medium
"*HD_DVD*"	*O*	HD DVD medium
Vendor-defined	*X*	

Table 2-10 — allowedValueList for *CurrentPlayMode*

Value	R/O
"*NORMAL*"	*R*
"*SHUFFLE*"	*O*
"*REPEAT_ONE*"	*O*
"*REPEAT_ALL*"	*O*
"*RANDOM*"	*O*
"*DIRECT_1*"	*O*
"*INTRO*"	*O*
Vendor-defined	

Table 2-11 — allowedValueList for *RecordMediumWriteStatus*

Value	R/O
"*WRITABLE*"	*O*
"*PROTECTED*"	*O*
"*NOT_WRITABLE*"	*O*
"*UNKNOWN*"	*O*
"*NOT_IMPLEMENTED*"	*O*
Vendor-defined	

Table 2-12 — allowedValueList for *CurrentRecordQualityMode*

Value	R/O
"*0:EP*"	*O*
"*1:LP*"	*O*
"*2:SP*"	*O*
"*0:BASIC*"	*O*
"*1:MEDIUM*"	*O*
"*2:HIGH*"	*O*
"*NOT_IMPLEMENTED*"	*O*
Vendor-defined	

Table 2-13 — allowedValueRange for *NumberOfTracks*

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

Table 2-14 — allowedValueRange for *CurrentTrack*

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

Table 2-15 — allowedValueList for *CurrentTransportActions*

Value	R/O
"*PLAY*"	*R*
"*STOP*"	*R*
"*PAUSE*"	*R*EQUIRED if *Pause()* action is implemented. Prohibited otherwise.
"*SEEK*"	*R*
"*NEXT*"	*R*
"*PREVIOUS*"	*R*
"*RECORD*"	*R*EQUIRED if *Record()* action is implemented. Prohibited otherwise.
Vendor-defined	

Table 2-16 — allowedValueList for *DRMState*

Value	R/O
"*OK*"	*R*
"*UNKNOWN*"	*O*
"*PROCESSING_CONTENT_KEY*"	*O*
"*CONTENT_KEY_FAILURE*"	*O*
"*ATTEMPTING_AUTHENTICATION*"	*O*
"*FAILED_AUTHENTICATION*"	*O*
"*NOT_AUTHENTICATED*"	*O*
"*DEVICE_REVOCATION*"	*O*

Table 2-17 — allowedValueList for *A_ARG_TYPE_SeekMode*

Value	R/O
"*TRACK_NR*"	*R*
"*ABS_TIME*"	*O*
"*REL_TIME*"	*O*
"*ABS_COUNT*"	*O*
"*REL_COUNT*"	*O*
"*CHANNEL_FREQ*"	*O*
"*TAPE-INDEX*"	*O*
"*REL_TAPE-INDEX*"	*O*
"*FRAME*"	*O*
"*REL_FRAME*"	*O*
Vendor-defined	

2.2.1 *TransportState*

This state variable forms the core of the AVTransport service. It defines the conceptually top-level state of the transport, for example, whether it is playing, recording, etc. Device vendors do not need to implement all allowed values of this variable, for example, non-recordable media will not implement the "*RECORDING*" state.

The "*PAUSED_RECORDING*" state is different from the "*STOPPED*" state in the sense that the transport is already prepared for recording and may respond faster or more accurate. The "*PAUSED_PLAYBACK*" state is different from the "*PAUSED_RECORDING*" state in the sense that in case the media contains video, it indicates output of a still image. The other *TransportState* values are self explanatory.

Note that *dubbing* of media at various speeds is not supported in this version of the AVTransport, mainly because there are no standards for cross-device dubbing speeds.

Device vendors are allowed to implement additional vendor-defined transport states. However, since the semantic meaning of these transport states is not specified, control points that find a AVTransport service in a transport state that they do not understand should refrain from interacting with that AVTransport service (for example, forcing the service into the "*STOPPED*" state). Rather, they should wait until the service transits back into a transport state that they understand.

2.2.2 *TransportStatus*

During operation of the AVTransport service, asynchronous errors may occur that cannot be returned by a normal action. For example, some time after playback of a stream has been started (via *SetAVTransportURI()* and *Play()* actions), there may be network congestion or server problems causing hiccups in the rendered media. These types of situations can be signaled to control points by setting this state variable to value "*ERROR_OCCURRED*". More specific error descriptions MAY also be used as vendor extensions. The value of *TransportState* after an error has occurred is implementation-dependent; some implementations MAY go to "*STOPPED*" while other implementations MAY be able to continue playing after an error. The time at which this state variable returns to "*OK*" after an error situation is also implementation dependent.

2.2.3 *CurrentMediaCategory*

This state variable indicates whether the current media is track-aware (both single and multi-track) or track-unaware (e.g. VHS-tape). The semantics of state variables *RelativeTimePosition*, *AbsoluteTimePosition*, *RelativeCounterPosition*, *AbsoluteCounterPosition* and of the *Seek()* action change, depending on this state variable.

2.2.4 *PlaybackStorageMedium*

This state variable indicates the storage medium of the resource specified by *AVTransportURI*. If no resource is specified, then the state variable is set to "*NONE*". If *AVTransportURI* refers to a resource received from the UPnP network, the state variable is set to "*NETWORK*". Device vendors MAY extend the specified allowed value list of this variable. For example, various types of solid-state media formats may be added in a vendor-specific way.

Note that this variable is not intended for signal- or content-formats such as MPEG2. Such type of information is exposed by the ConnectionManager service associated with this service.

2.2.5 *RecordStorageMedium*

This state variable indicates the storage medium where the resource specified by *AVTransportURI* will be recorded when a Record action is issued. If no resource is specified, then the state variable is set to "*NONE*". Device vendors MAY extend the allowed value list of this variable. For example, various types of solid-state media formats may be added in a vendor-specific way.

Note that this variable is not intended for signal- or content-formats such as MPEG2. Such type of information is exposed by the ConnectionManager service associated with this service. If the service implementation does not support recording, then this state variable MUST be set to "*NOT_IMPLEMENTED*".

2.2.6 *PossiblePlaybackStorageMedia*

This state variable contains a static, comma-separated list of storage media that the device can play. RECOMMENDED values are defined in the allowed value list for state variable *PlaybackStorageMedium*.

2.2.7 *PossibleRecordStorageMedia*

This state variable contains a static, comma-separated list of storage media onto which the device can record. RECOMMENDED values are defined in the allowed value list for state variable *RecordStorageMedium*. If the service implementation does not support recording, then this state variable MUST be set to "*NOT_IMPLEMENTED*".

2.2.8 *CurrentPlayMode*

This state variable indicates the current play mode (for example, random play, repeated play, etc.). This notion is typical for CD-based audio media, but is generally not supported by tape-based media. Value "*DIRECT_1*" indicates playing a single track and then stop (don't play the next track). Value "*INTRO*" indicates playing a short sample (typically 10 seconds or so) of each track on the media. Other play mode values are self explanatory.

2.2.9 *TransportPlaySpeed*

This state variable is a string representation of a rational fraction that indicates the speed relative to normal speed. Example values are "1", "1/2", "2", "-1", "1/10", etc. Actually supported speeds can be retrieved from the AllowedValueList of this state variable in the AVTransport service description. Value "1" is REQUIRED, value "0" is not allowed. Negative values indicate reverse playback.

2.2.10 *RecordMediumWriteStatus*

This state variable reflects the write protection status of the currently loaded media. "*NOT_WRITABLE*" indicates an inherent *read-only* media (for example, a DVD-ROM disc) or the device doesn't support recording on the current media. "*PROTECTED*" indicates a writable media that is currently write-protected (for example, a protected VHS tape). If no media is loaded, the write status will be "*UNKNOWN*". If the service implementation does not support recording, then this state variable MUST be set to "*NOT_IMPLEMENTED*".

2.2.11 *CurrentRecordQualityMode*

This state variable indicates the currently set record quality mode. Such a setting takes the form of "Quality Ordinal:label". The Quality Ordinal indicates a particular relative quality level available in the device, from 0 (lowest quality) to n (highest quality). The label associated with the ordinal provides a human-readable indication of the ordinal's meaning. If the service implementation does not support recording, then this state variable MUST be set to "*NOT_IMPLEMENTED*".

2.2.12 *PossibleRecordQualityModes*

This state variable contains a static, comma-separated list of recording quality modes that the device supports. For example, for an analog VHS recorder the string would be "*0:EP,1:LP,2:SP*", while for a PVR the string would be "*0:BASIC,1:MEDIUM,2:HIGH*". The string specifics depend on the type of device containing the AVTransport. Note that record quality modes are independent of the *content-format* that may be exposed to the network through a ConnectionManager service. If the service implementation does not support recording, then this state variable MUST be set to "*NOT_IMPLEMENTED*".

2.2.13 *NumberOfTracks*

This state variable contains the number of tracks controlled by the AVTransport instance. If no resource is associated with the AVTransport instance (via *SetAVTransportURI()*), and there is no *default* resource (for example, a loaded disc) then *NumberOfTracks* MUST be 0. Also, if the implementation is never able to determine the number of tracks in the currently selected media, *NumberOfTracks* MUST be set to 0. Otherwise, it MUST be 1 or higher. In some cases, for example, large playlist, it may take a long time to determine the exact number of tracks. Until the exact number is determined, the value of the state variable is implementation dependent, for example, keeping it to 1 until determined or updating the value periodically. Note that in any case, the AVTransport service MUST generate a *LastChange* event with defined moderation period when the exposed value is updated.

For track-unaware media, this state variable will always be set to 1. For LD and DVD media, a track is defined as a chapter number. For Tuners that provide an indexed list of channels, a track is defined as an index number in such a list. This state variable has to be consistent with the resource identified by *AVTransportURI*. For example, if *AVTransportURI* points to a single MP3 file, then *NumberOfTracks* MUST be set to 1. However, if *AVTransportURI* points to a playlist file, then *NumberOfTracks* MUST be equal to the number of entries in the playlist.

2.2.14 *CurrentTrack*

If *NumberOfTracks* is 0, then *CurrentTrack* will be 0. Otherwise, this state variable contains the sequence number of the currently selected track, starting at value 1, up to and including *NumberOfTracks*. For track-unaware media, this state variable is always 1. For LD and DVD media, the notion of track equals the notion of chapter number. For Tuners that provide an indexed list of channels, the current track is defined as the current index number in such a list.

2.2.15 *CurrentTrackDuration*

This state variable contains the duration of the current track, specified as a string of the following form:

H+:MM:SS[.F+] or H+:MM:SS[.F0/F1]

where:

- H+ means one or more digits to indicate elapsed hours
- MM means exactly 2 digits to indicate minutes (00 to 59)
- SS means exactly 2 digits to indicate seconds (00 to 59)
- [.F+] means OPTIONALLY a dot followed by one or more digits to indicate fractions of seconds
- [.F0/F1] means OPTIONALLY a dot followed by a fraction, with F0 and F1 at least one digit long, and F0 < F1

The string MAY be preceded by an OPTIONAL + or – sign, and the decimal point itself MUST be omitted if there are no fractional second digits. This variable does not apply to Tuners. If the implementation is never able to determine the duration of the current track, *CurrentTrackDuration* MUST be set to “00:00:00”. If the optional fractional components are included, they MUST be set to either “0” or “0/<F1>”. In some cases, it may take a long time to determine the exact duration of tracks. Until the exact duration is determined, the value of the state variable is implementation dependent, for example, keeping it to “00:00:00” or updating the value periodically. Note that in any case, the AVTransport service MUST generate a *LastChange* event with defined moderation period when the exposed value is updated. If the service implementation does not support track duration information then this state variable MUST be set to “*NOT_IMPLEMENTED*”.

2.2.16 *CurrentMediaDuration*

This state variable contains the duration of the media, as identified by state variable *AVTransportURI*. In case the *AVTransportURI* represents only 1 track, this state variable is equal to *CurrentTrackDuration*. The format of this variable is the same as the format for *CurrentTrackDuration*, described above. If no content is associated with the AVTransport instance (via *SetAVTransportURI()*), and there is no default content (for example, a loaded disc) then *CurrentMediaDuration* MUST be set to "*00:00:00*". Also, if the implementation is never able to determine the duration of the currently selected media, *CurrentMediaDuration* MUST be set to "00:00:00". If the optional fractional components are included, they MUST be set to either "0" or "0/<F1>". In some cases, it may take a long time to determine the exact duration of the media. Until the exact duration is determined, the value of the state variable is implementation dependent, for example, keeping it to "00:00:00" or updating the value periodically. Note that in any cases, the AVTransport service MUST generate a *LastChange* event with defined moderation period when the exposed value is updated. If the service implementation does not support media duration information, then this state variable MUST be set to "*NOT_IMPLEMENTED*".

2.2.17 *CurrentTrackMetaData*

This state variable contains the meta-data, in the form of a *DIDL-Lite XML Fragment* (defined in the ContentDirectory service template), associated with the resource pointed to by state variable *CurrentTrackURI*. The meta-data may have been extracted from state variable *AVTransportURIMetaData*, or extracted from the resource binary itself (for example, embedded ID3 tags for MP3 audio). This is implementation dependent. If the service implementation does not support this feature, then this state variable MUST be set to "*NOT_IMPLEMENTED*".

2.2.18 *CurrentTrackURI*

This state variable contains a reference, in the form of a URI, to the current track. The URI may enable a control point to retrieve any meta-data associated with the current track, such as title and author information, via the ContentDirectory service *Browse()* and/or *Search()* action. In case the media does contain multi-track content, but there is no separate URI associated with each track, *CurrentTrackURI* MUST be set equal to *AVTransportURI*.

2.2.19 *AVTransportURI*

This state variable contains a reference, in the form of a URI, to the resource controlled by the AVTransport instance. This URI may refer to a single item (for example, a song) or to a collection of items (for example, a playlist). In the *single item* case, the AVTransport will have 1 track and *AVTransportURI* is equal to *CurrentTrackURI*. In the *collection of items* case, the AVTransport will have multiple tracks, and *AVTransportURI* will remain constant during track changes. The URI enables a control point to retrieve any meta-data associated with the AVTransport instance, such as title and author information, via the ContentDirectory service.

2.2.20 *AVTransportURIMetaData*

This state variable contains the meta-data, in the form of a *DIDL-Lite XML Fragment* (defined in the ContentDirectory service template), associated with the resource pointed to by state variable *AVTransportURI*. See the ContentDirectory service specification for details. If the service implementation does not support this feature, then this state variable MUST be set to "*NOT_IMPLEMENTED*".

2.2.21 *NextAVTransportURI*

This state variable contains the *AVTransportURI* value to be played when the playback of the current *AVTransportURI* finishes. Setting this variable ahead of time (via action *SetNextAVTransportURI()*) enables a device to provide seamless transitions between resources for certain data transfer protocols that need buffering (for example, HTTP). If the

service implementation does not support this feature, then this state variable MUST be set to "*NOT_IMPLEMENTED*".

Do not confuse transitions between *AVTransportURI* and *NextAVTransportURI* with track transitions. When *AVTransportURI* is set to a playlist, *NextAVTransportURI* will be played when the whole playlist finishes, not when the current playlist entry (*CurrentTrackURI*) finishes.

2.2.22 *NextAVTransportURIMetaData*

This state variable contains the meta-data, in the form of a *DIDL-Lite XML Fragment* (defined in the ContentDirectory service template), associated with the resource pointed to by state variable *NextAVTransportURI*. See the ContentDirectory service specification for details. If the service implementation does not support this feature then this state variable MUST be set to "*NOT_IMPLEMENTED*".

2.2.23 *RelativeTimePosition*

For track-aware media, this state variable contains the current position in the current track, in terms of time, measured from the beginning of the current track. The range for this state variable is from "00:00:00" to the duration of the current track as indicated by the *CurrentTrackDuration* state variable. For track-aware media, this state variable always contains a positive value.

For track-unaware media (e.g. a single tape), this state variable contains the position, in terms of time, measured from a *zero reference point* on the media. The range for this state variable is from the beginning of the media, measured from the zero reference point to the end of the media, also measured from the zero reference point. For track-unaware media, this state variable can be negative. Indeed, when the zero reference point does not coincide with the beginning of the media, all positions before the zero reference point are expressed as negative values.

The time format used for the *RelativeTimePosition* state variable is the same as for state variable *CurrentTrackDuration*. If the service implementation does not support relative time-based position information, then this state variable MUST be set to *"NOT_IMPLEMENTED"*.

2.2.24 *AbsoluteTimePosition*

This state variable contains the current position, in terms of time, measured from the beginning of the media. The time format used for the *AbsoluteTimePosition* state variable is the same as for state variable *CurrentTrackDuration*. The range for this state variable is from "00:00:00" to the duration of the current media as indicated by the *CurrentMediaDuration* state variable. This state variable always contains a positive value.

If the service implementation does not support any kind of position information, then this state variable MUST be set to *"NOT_IMPLEMENTED"*. Devices that do not have time position information, but are able to detect whether they are at the end of the media MUST use special value *"END_OF_MEDIA"* when actually at the end, and the value *"NOT_IMPLEMENTED"* otherwise.

2.2.25 *RelativeCounterPosition*

For track-aware media, this state variable contains the current position in the current track, in terms of a dimensionless counter, measured from the beginning of the current track. The range for this state variable is from 0 to the counter value that corresponds to the end of the current track. For track-aware media, this state variable always contains a positive value.

For track-unaware media (e.g. a single tape), this state variable contains the position, in terms of a dimensionless counter, measured from a *zero reference point* on the media. The

range for this state variable is from the counter value that corresponds to the beginning of the media, measured from the zero reference point to the counter value that corresponds to the end of the media, also measured from the zero reference point. For track-unaware media, this state variable can be negative, Indeed, when the zero reference point does not coincide with the beginning of the media, all positions before the zero reference point are expressed as negative values.

For devices that support media with addressable ranges that equal or exceed the allowed range of this counter, the AVTransport service must scale actual media addresses to counter values to fit within the range allowed for this counter.

If the service implementation does not support relative count-based position information, then this state variable MUST be set to the maximum value of the **i4** data type.

2.2.26 *AbsoluteCounterPosition*

This state variable contains the current position, in terms of a dimensionless counter, measured from the beginning of the loaded media. The allowed range for this variable is [0, 2147483646]. For devices that support media with addressable ranges that equal or exceed the allowed range of this counter, the AVTransport service must scale actual media addresses to counter values to fit within the range allowed for this counter. If the service implementation does not support absolute count-based position information, then this state variable MUST be set to the value 2147483647.

Note: Although the data type for state variable *AbsoluteCounterPosition* is **ui4**, the range is restricted to [0, Max(**i4**)] for backwards compatibility reasons.

2.2.27 *CurrentTransportActions*

This state variable contains a comma-separated list of transport-controlling actions that can be successfully invoked for the current resource at this specific point in time. The list MUST contain a subset (including the empty set) of the following action names: "*Play*", "*Stop*", "*Pause*", "*Seek*", "*Next*", "*Previous*" and "*Record*". In addition, the list MAY be augmented by a subset of vendor-defined transport-controlling action names. For example, when a live stream from the Internet is being controlled, the variable may be only "*Play,Stop*". When a local audio CD is being controlled, the variable may be "*Play,Stop,Pause,Seek,Next,Previous*". This information can be used, for example, to dynamically enable or disable play, stop, pause buttons, etc., on a user interface.

2.2.28 *LastChange*

This state variable is used for eventing purposes to allow clients to receive meaningful event notifications whenever the state of the AVTransport changes. Logically, it contains a list of pairs, one element being an AVTransport instance ID and the second element the name and new value of the state variable for that instance. The format of the *LastChange* state variable is defined in [AVT-EVENT-XSD]. The *LastChange* state variable follows the behavior of the *LastChange* state variable as described in the RenderingControl Service Specification [RCS], Clause 2.2.1, "*LastChange*".

2.2.29 *DRMState*

The state variable *DRMState* is required for the AVTransport service on AV Media Renderers and Media Servers that implement the AVTransport service for the purpose of controlling the transport of DRM-controlled content. Correspondingly, the *DRMState* state variable must not be included in AVTransport service implementations that do no control the transport of DRM-controlled content.

The *DRMState* state variable is used by instances of the AVTransport service to inform control points about process failures and other AVTransport instance state changes that may occur independently of AVTransport actions. An authentication failure for example, may be

detected well after the completion of the *SetAVTransportURI()* action that triggered the authentication process. In this case, the *DRMState* state variable is set to "*FAILED_AUTHENTICATION*" and evented to inform the control point.

The detailed state-transition diagram associated with the values of the *DRMState* state variable is implementation-dependent. It is highly recommended that AVTransport service implementations adhere to the following recommendations for setting *DRMState* values in order to maintain maximum interoperability:

- The *DRMState* state variable is set to "*NOT_AUTHENTICATED*" upon AVTransport instance initialization, allowing the AVTransport instance to indicate to the control point that authentication is both required and has not yet occurred. The value of the *DRMState* state variable is also set to "*NOT_AUTHENTICATED*" to indicate an expiration, time-out, or some other condition that results in a transition to a non-authenticated state.
- The *DRMState* state variable is set to "*ATTEMPTING_AUTHENTICATION*" when the AVTransport instance detects that an authentication process is occurring.
- The *DRMState* state variable is set to "*FAILED_AUTHENTICATION*" following "*ATTEMPTING_AUTHENTICATION*" to indicate that the AVTransport instance has detected that the authentication process has failed.
- The *DRMState* state variable is set to "*CONTENT_KEY_FAILURE*" to inform the control point that a content key needed to either start or continue a media transport state was either not received or has failed verification.
- The *DRMState* state variable is set to "*DEVICE_REVOCATION*" specifically to inform the control point that the AVTransport instance has detected a revocation condition.

The other *DRMState* values are self explanatory.

The optional action *GetDRMState()* is required to be implemented when the *DRMState* state variable is implemented.

2.2.30 *A_ARG_TYPE_SeekMode*

This state variable is introduced to provide type information for the *Unit* argument in action *Seek()*. It indicates the allowed units in which the amount of seeking to be performed is specified. It can be specified as a time (relative or absolute), a count (relative or absolute), a track number, a tape-index (for example, for tapes with an indexing facility; relative or absolute) or even a video frame (relative or absolute). A device vendor is allowed to implement a subset of the allowed value list of this state variable. Only the value "*TRACK_NR*" is REQUIRED.

2.2.31 *A_ARG_TYPE_SeekTarget*

This state variable is introduced to provide type information for the *Target* argument in action *Seek()*. It indicates the target position of the *Seek()* action, in terms of units defined by state variable *A_ARG_TYPE_SeekMode*. The data type of this variable is **string**. However, depending on the actual seek mode used, it MUST contain string representations of values as defined in the following table:

Table 2-18 — Format of *A_ARG_TYPE_SeekTarget*

Value of *A_ARG_TYPE_SeekMode*	Format of *A_ARG_TYPE_SeekTarget*
"*TRACK_NR*"	**ui4**
"*ABS_TIME*"	Formatted as specified in Clause 2.2.14, "*CurrentTrackDuration*"
"*REL_TIME*"	Formatted as specified in Clause 2.2.14, "*CurrentTrackDuration*"
"*ABS_COUNT*"	**ui4**
"*REL_COUNT*"	**i4**
"*CHANNEL_FREQ*"	**float**, expressed in Hz.
"*TAPE-INDEX*"	**ui4**
"*REL_TAPE-INDEX*"	**i4**
"*FRAME*"	**ui4**
"*REL_FRAME*"	**i4**

Supported ranges of these integer, time or float values are device-dependent.

2.2.32 *A_ARG_TYPE_InstanceID*

This state variable is introduced to provide type information for the *InstanceID* input argument present in all AVTransport actions. It identifies the virtual instance of the AVTransport service to which the action applies. A valid *InstanceID* is obtained from a *factory* method in the ConnectionManager service: the *ConnectionManager::PrepareForConnection()* action.

If the device's ConnectionManager does not implement the optional *ConnectionManager::PrepareForConnection()* action, special value "*0*" MUST be used for the *InstanceID* input argument. In such a case, the device implements a single static AVTransport instance, and only one stream can be controlled and sent (or received) at any time.

2.2.33 *A_ARG_TYPE_DeviceUDN*

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

2.2.34 *A_ARG_TYPE_ServiceType*

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

2.2.35 *A_ARG_TYPE_ServiceID*

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

2.2.36 *A_ARG_TYPE_StateVariableValuePairs*

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

The following *stateVariableValuePairs XML Document* 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.xsd">
   <stateVariable variableName="CurrentPlayMode">
      NORMAL
   </stateVariable>
   <stateVariable variableName="CurrentTrack">
      3
   </stateVariable>
   <!—- More state variable value pairs can be inserted here -->
</stateVariableValuePairs>
```

The relevant variable names MUST be either all or a subset (as required) of the defined AVTransport state variables except for *LastChange* and any *A_ARG_TYPE_xxx* state variables.

2.2.37 *A_ARG_TYPE_StateVariableList*

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

2.3 Eventing and Moderation

Table 2-19 — Event Moderation

Variable Name	Evented	Moderated Event	Max Event Rate [a]	Logical Combination	Min Delta per Event [b]
TransportState	NO	NO			
TransportStatus	NO	NO			
CurrentMediaCategory	NO	NO			
PlaybackStorageMedium	NO	NO			
PossiblePlaybackStorageMedia	NO	NO			
PossibleRecordStorageMedia	NO	NO			
CurrentPlayMode	NO	NO			
TransportPlaySpeed	NO	NO			
RecordMediumWriteStatus	NO	NO			
PossibleRecordQualityModes	NO	NO			
CurrentRecordQualityMode	NO	NO			
NumberOfTracks	NO	NO			
CurrentTrack	NO	NO			
CurrentTrackDuration	NO	NO			
CurrentMediaDuration	NO	NO			
CurrentTrackURI	NO	NO			
CurrentTrackMetaData	NO	NO			
AVTransportURI	NO	NO			
AVTransportURIMetaData	NO	NO			
NextAVTransportURI	NO	NO			
NextAVTransportURIMetaData	NO	NO			
CurrentTransportActions	NO	NO			
LastChange	YES	YES	0.2 seconds		
DRMState	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			

a Determined by N, where Rate = (Event)/(N secs).

b (N) * (allowedValueRange Step).

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

2.3.1 Event Model

Since the AVTransport 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

AVTransport 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, except the position-related state variables listed below, are indirectly evented via the *LastChange* state variable. (Note: A_ARG_TYPE_ state variables are not evented, either directly or indirectly.). More details about the *LastChange*-based event mechanism can be found in the Event Model clause of the RenderingControl service.

The AVTransport service contains various state variables that, during certain transport states, change almost continuously. The following variables are therefore not evented via *LastChange*:

- *RelativeTimePosition*
- *AbsoluteTimePosition*
- *RelativeCounterPosition*
- *AbsoluteCounterPosition*

Each control point can poll for these values at a rate appropriate for their application, whenever they need to. For example, a control point can invoke *GetPositionInfo()* every second when the *TransportState* is "*PLAYING*", "*RECORDING*" or "*TRANSITIONING*". This is more efficient and flexible than requiring event notifications to be sent to all subscribing control points, in all cases.

Evented state variables MUST only be evented if their value actually changes. Writing the same value to a state variable does not generate an event. For example, a transition from the state "*PLAYING*" to the state "*PLAYING*"with a different speed does not generate an event for state variable *TransportState* ("*PLAYING*" → "*PLAYING*"). However, this transition will generate an event for the state variable *TransportPlaySpeed*. If a moderated state variable is evented and it returns the same value, this means that within the moderation time, its value has actually changed and then changed back to its previous value.

2.4 Actions

Table 2-20 — Actions

Name	R/O [a]
SetAVTransportURI()	*R*
SetNextAVTransportURI()	*O*
GetMediaInfo()	*R*
GetMediaInfo_Ext()	*R*
GetTransportInfo()	*R*
GetPositionInfo()	*R*
GetDeviceCapabilities()	*R*
GetTransportSettings()	*R*
Stop()	*R*
Play()	*R*
Pause()	*O*
Record()	*O*
Seek()	*R*
Next()	*R*
Previous()	*R*
SetPlayMode()	*O*
SetRecordQualityMode()	*O*
GetCurrentTransportActions()	*O*
GetDRMState()	*O* [b]
GetStateVariables()	*O*
SetStateVariables()	*O*
Non-standard actions implemented by a UPnP vendor go here	*X*

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

[b] REQUIRED if the *DRMState* state variable is implemented

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

2.4.1 *SetAVTransportURI()*

This action specifies the URI of the resource to be controlled by the specified AVTransport instance. It is RECOMMENDED that the AVTransport service checks the MIME-type of the specified resource when executing this action. For AVTransport instances that control the transport of DRM-controlled content, the authentication process is also RECOMMENDED to start as a result of executing this action. The *SetAVTransportURI()* action is successful even when a required authentication or revocation check cannot be completed before the expiration of time allotted for the completion of the *SetAVTransportURI()* action. In the case of AVTransport instances that control the transport of DRM-controlled content, the subsequent detection of conditions that need to be communicated to the control point, like an authentication failure, a revocation condition, etc. are indicated via the *DRMState* state variable. A control point can supply metadata associated with the specified resource, using a *DIDL-Lite XML Fragment* (defined in the ContentDirectory service specification), in argument *CurrentURIMetaData*. If supported by the AVTransport instance, this metadata is stored in a state variable, and returned as output argument as part of the *GetMediaInfo()* action. If a control point does not want to use this feature it can supply the empty string for the *CurrentURIMetaData* argument.

2.4.1.1 Arguments

Table 2-21 — Arguments for *SetAVTransportURI()*

Argument	Direction	relatedStateVariable
InstanceID	*IN*	*A_ARG_TYPE_InstanceID*
CurrentURI	*IN*	*AVTransportURI*
CurrentURIMetaData	*IN*	*AVTransportURIMetaData*

2.4.1.2 Dependency on State

None.

2.4.1.3 Effect on State

Depending on the URI, the number of tracks available on this instance may have changed. For example, if the URI points to a single audio file, state variable *NumberOfTracks* changes to 1. However, if the URI points to an audio playlist, state variable *NumberOfTracks* changes to the number of entries in the playlist.

If the renderer fails to locate or download the resource at the URI the *TransportState* MUST change to "*STOPPED*". If the current transport state is "*PLAYING*", and it would take a noticeable amount of time before a human user would actually see or hear the media at the new URI playing, the AVTransport is allowed to temporarily go to the "*TRANSITIONING*" state before going back to "*PLAYING*". This might be appropriate for devices that need to start buffering or completely download the media before playback can start. If the current transport state is "*NO MEDIA PRESENT*" the transport state changes to "*STOPPED*". In all other cases, this action does not change the transport state of the specified instance.

2.4.1.4 Errors

Table 2-22 — Error Codes for *SetAVTransportURI()*

ErrorCode	errorDescription	Description
400-499	TBD	See UPnP Device Architecture clause on Control.
500-599	TBD	See UPnP Device Architecture clause on Control.
600-699	TBD	See UPnP Device Architecture clause on Control.
714	Illegal MIME-type	The specified resource has a MIME-type which is not supported by the AVTransport service.
715	Content 'BUSY'	This indicates that the resource is already in use at this time.
716	Resource not found	The specified resource cannot be found in the network.
718	Invalid InstanceID	The specified *InstanceID* is invalid for this AVTransport.
719	DRM error	The action failed because an unspecified DRM error occurred.
720	Expired content	The action failed because the content use validity interval has expired.
721	Non-allowed use	The action failed because the requested content use is disallowed.
722	Can't determine allowed uses	The action failed because the allowed content uses cannot be verified.
723	Exhausted allowed use	The action failed because the number of times this content has been used as requested has reached the maximum allowed number of uses.
724	Device authentication failure	The action failed because of a device authentication failure between the media source device and the media sink device.
725	Device revocation	The action failed because either the media source device or the media sink device has been revoked.

2.4.2 *SetNextAVTransportURI()*

This action specifies the URI of the resource to be controlled when the playback of the current resource (set earlier via *SetAVTransportURI()*) finishes. This action allows a device to *prefetch* the data to be played next, in order to provide a seamless transition between resources. This type of prefetching or buffering is particularly useful for protocols such as HTTP, where the data is usually buffered before playback. It is RECOMMENDED that the AVTransport service checks the MIME-type of the specified resource when executing this action. For AVTransport instances that control the transport of DRM-controlled content, the authentication process is also RECOMMENDED to start as a result of executing this action. The *SetNextAVTransportURI()* action is successful even when a required authentication or revocation check cannot be completed before the expiration of time allotted for the completion of the *SetNextAVTransportURI()* action. In the case of AVTransport instances that control the transport of DRM-controlled content, the subsequent detection of conditions that need to be communicated to the control point, like an authentication failure, a revocation condition, etc. are indicated via the *DRMState* state variable.

A control point can supply meta-data, using a *DIDL-Lite XML Fragment* (defined in the ContentDirectory service specification), via argument *NextURIMetaData*. If supported by the AVTransport service, this meta-data is stored in a state variable, and returned as an output argument as part of action *GetMediaInfo()*. If a control point does not want to use this feature it can supply the empty string for the *NextURIMetaData* argument.

2.4.2.1 Arguments

Table 2-23 — Arguments for *SetNextAVTransportURI()*

Argument	Direction	relatedStateVariable
InstanceID	*IN*	*A_ARG_TYPE_InstanceID*
NextURI	*IN*	*NextAVTransportURI*
NextURIMetaData	*IN*	*NextAVTransportURIMetaData*

2.4.2.2 Dependency on State

None.

2.4.2.3 Effect on State

This action does **not** change the transport state of the specified instance. In case that the next URI buffer exists (that is: a legal URI which will be rendered next has been located), when the playback of the current resource finishes, state variable *AVTransportURI* changes to the value of state variable *NextAVTransportURI*. The same holds for *AVTransportURIMetaData* and *NextAVTransportURIMetaData*. The process repeats itself until there is no more URI to be rendered. In such case, the state variable *NextAVTransportURI* will be set to the empty string.

If an illegal URI is used for the *SetNextAVTransportURI()* action, which is detected immediately and most likely while the current URI is still being rendered, the current transport state MUST be kept. After the current URI finishes playing, the transition to that illegal URI cannot be made and *TransportState* MUST be set to "*STOPPED*".

2.4.2.4 Errors

Table 2-24 — Error Codes for *SetNextAVTransportURI()*

ErrorCode	errorDescription	Description
400-499	TBD	See UPnP Device Architecture clause on Control.
500-599	TBD	See UPnP Device Architecture clause on Control.
600-699	TBD	See UPnP Device Architecture clause on Control.
714	Illegal MIME-type	The specified resource has a MIME-type which is not supported by the AVTransport service.
715	Content 'BUSY'	This indicates that the resource is already in use at this time.
716	Resource not found	The specified resource cannot be found in the network.
718	Invalid InstanceID	The specified *InstanceID* is invalid for this AVTransport.
719	DRM error	The action failed because an unspecified DRM error occurred.
720	Expired content	The action failed because the content use validity interval has expired.
721	Non-allowed use	The action failed because the requested content use is disallowed.
722	Can't determine allowed uses	The action failed because the allowed content uses cannot be verified.
723	Exhausted allowed use	The action failed because the number of times this content has been used as requested has reached the maximum allowed number of uses.
724	Device authentication failure	The action failed because of a device authentication failure between the media source device and the media sink device.
725	Device revocation	The action failed because either the media source device or the media sink device has been revoked.

2.4.3 *GetMediaInfo()*

This action returns information associated with the current media of the specified instance; it has no effect on state.

2.4.3.1 Arguments

Table 2-25 — Arguments for *GetMediaInfo()*

Argument	Direction	relatedStateVariable
InstanceID	*IN*	*A_ARG_TYPE_InstanceID*
NrTracks	*OUT*	*NumberOfTracks*
MediaDuration	*OUT*	*CurrentMediaDuration*
CurrentURI	*OUT*	*AVTransportURI*
CurrentURIMetaData	*OUT*	*AVTransportURIMetaData*
NextURI	*OUT*	*NextAVTransportURI*
NextURIMetaData	*OUT*	*NextAVTransportURIMetaData*
PlayMedium	*OUT*	*PlaybackStorageMedium*
RecordMedium	*OUT*	*RecordStorageMedium*
WriteStatus	*OUT*	*RecordMediumWriteStatus*

2.4.3.2 Dependency on State

None.

2.4.3.3 Effect on State

None.

2.4.3.4 Errors

Table 2-26 — Error Codes for *GetMediaInfo()*

ErrorCode	errorDescription	Description
400-499	TBD	See UPnP Device Architecture clause on Control.
500-599	TBD	See UPnP Device Architecture clause on Control.
600-699	TBD	See UPnP Device Architecture clause on Control.
718	Invalid InstanceID	The specified *InstanceID* is invalid for this AVTransport.

2.4.4 *GetMediaInfo_Ext()*

This action returns information associated with the current media of the specified instance; it has no effect on state.The information returned is identical to the information returned by the *GetMediaInfo()* action, except for the additionally returned *CurrentType* argument

2.4.4.1 Arguments

Table 2-27 — Arguments for *GetMediaInfo_Ext()*

Argument	Direction	relatedStateVariable
InstanceID	*IN*	*A_ARG_TYPE_InstanceID*
CurrentType	*OUT*	*CurrentMediaCategory*
NrTracks	*OUT*	*NumberOfTracks*
MediaDuration	*OUT*	*CurrentMediaDuration*
CurrentURI	*OUT*	*AVTransportURI*
CurrentURIMetaData	*OUT*	*AVTransportURIMetaData*
NextURI	*OUT*	*NextAVTransportURI*
NextURIMetaData	*OUT*	*NextAVTransportURIMetaData*
PlayMedium	*OUT*	*PlaybackStorageMedium*
RecordMedium	*OUT*	*RecordStorageMedium*
WriteStatus	*OUT*	*RecordMediumWriteStatus*

2.4.4.2 Dependency on State

None.

2.4.4.3 Effect on State

None.

2.4.4.4 Errors

Table 2-28 — Error Codes for *GetMediaInfo_Ext()*

ErrorCode	errorDescription	Description
400-499	TBD	See UPnP Device Architecture clause on Control.
500-599	TBD	See UPnP Device Architecture clause on Control.
600-699	TBD	See UPnP Device Architecture clause on Control.
718	Invalid InstanceID	The specified *InstanceID* is invalid for this AVTransport.

2.4.5 *GetTransportInfo()*

This action returns information associated with the current transport state of the specified instance; it has no effect on state.

2.4.5.1 Arguments

Table 2-29 — Arguments for *GetTransportInfo()*

Argument	Direction	relatedStateVariable
InstanceID	*IN*	*A_ARG_TYPE_InstanceID*
CurrentTransportState	*OUT*	*TransportState*
CurrentTransportStatus	*OUT*	*TransportStatus*
CurrentSpeed	*OUT*	*TransportPlaySpeed*

2.4.5.2 Dependency on State

None.

2.4.5.3 Effect on State

None.

2.4.5.4 Errors

Table 2-30 — Error Codes for *GetTransportInfo()*

ErrorCode	errorDescription	Description
400-499	TBD	See UPnP Device Architecture clause on Control.
500-599	TBD	See UPnP Device Architecture clause on Control.
600-699	TBD	See UPnP Device Architecture clause on Control.
718	Invalid InstanceID	The specified *InstanceID* is invalid for this AVTransport.

2.4.6 *GetPositionInfo()*

This action returns information associated with the current position of the transport of the specified instance; it has no effect on state.

2.4.6.1 Arguments

Table 2-31 — Arguments for *GetPositionInfo()*

Argument	Direction	relatedStateVariable
InstanceID	*IN*	*A_ARG_TYPE_InstanceID*
Track	*OUT*	*CurrentTrack*
TrackDuration	*OUT*	*CurrentTrackDuration*
TrackMetaData	*OUT*	*CurrentTrackMetaData*
TrackURI	*OUT*	*CurrentTrackURI*
RelTime	*OUT*	*RelativeTimePosition*
AbsTime	*OUT*	*AbsoluteTimePosition*
RelCount	*OUT*	*RelativeCounterPosition*
AbsCount	*OUT*	*AbsoluteCounterPosition*

2.4.6.2 Dependency on State

None.

2.4.6.3 Effect on State

None.

2.4.6.4 Errors

Table 2-32 — Error Codes for *GetPositionInfo()*

ErrorCode	errorDescription	Description
400-499	TBD	See UPnP Device Architecture clause on Control.
500-599	TBD	See UPnP Device Architecture clause on Control.
600-699	TBD	See UPnP Device Architecture clause on Control.
718	Invalid InstanceID	The specified *InstanceID* is invalid for this AVTransport.

2.4.7 *GetDeviceCapabilities()*

This action returns information on device capabilities of the specified instance, such as the supported playback and recording formats, and the supported quality levels for recording. This action has no effect on state.

2.4.7.1 Arguments

Table 2-33 — Arguments for *GetDeviceCapabilities()*

Argument	Direction	relatedStateVariable
InstanceID	*IN*	*A_ARG_TYPE_InstanceID*
PlayMedia	*OUT*	*PossiblePlaybackStorageMedia*
RecMedia	*OUT*	*PossibleRecordStorageMedia*
RecQualityModes	*OUT*	*PossibleRecordQualityModes*

2.4.7.2 Dependency on State

None.

2.4.7.3 Effect on State

None.

2.4.7.4 Errors

Table 2-34 — Error Codes for *GetDeviceCapabilities()*

ErrorCode	errorDescription	Description
400-499	TBD	See UPnP Device Architecture clause on Control.
500-599	TBD	See UPnP Device Architecture clause on Control.
600-699	TBD	See UPnP Device Architecture clause on Control.
718	Invalid InstanceID	The specified *InstanceID* is invalid for this AVTransport.

2.4.8 *GetTransportSettings()*

This action returns information on various settings of the specified instance, such as the current play mode and the current recording quality mode. This action has no effect on state.

2.4.8.1 Arguments

Table 2-35 — Arguments for *GetTransportSettings()*

Argument	Direction	relatedStateVariable
InstanceID	*IN*	*A_ARG_TYPE_InstanceID*
PlayMode	*OUT*	*CurrentPlayMode*
RecQualityMode	*OUT*	*CurrentRecordQualityMode*

2.4.8.2 Dependency on State

None.

2.4.8.3 Effect on State

None.

2.4.8.4 Errors

Table 2-36 — Error Codes for *GetTransportSettings()*

ErrorCode	errorDescription	Description
400-499	TBD	See UPnP Device Architecture clause on Control.
500-599	TBD	See UPnP Device Architecture clause on Control.
600-699	TBD	See UPnP Device Architecture clause on Control.
718	Invalid InstanceID	The specified *InstanceID* is invalid for this AVTransport.

2.4.9 *Stop()*

This action stops the progression of the current resource that is associated with the specified instance. Additionally, it is RECOMMENDED that the *output of the device* (defined below) SHOULD change to something other than the current snippet of resource. Although the exact nature of this change varies from device to device, a common behavior is to immediately cease all output from the device. Nevertheless, the exact behavior is defined by the manufacturer of the device.

On some devices, the current position on the transport changes as a result of the *Stop()* action. This can be detected by control points via event notification of state variable *CurrentTrack*. Alternatively, a control point can poll using the *GetPositionInfo()* action.

Output of a device: In this context, the term *output of the device* (used above) has different semantics depending on the type of device that has implemented this AVTransport service. Some devices (for example, MediaServer devices) output media content to the network while other devices (for example, a MediaRenderer) output a visual and/or audio representation of media content that was received from the network.

2.4.9.1 Arguments

Table 2-37 — Arguments for *Stop()*

Argument	Direction	relatedStateVariable
InstanceID	*IN*	*A_ARG_TYPE_InstanceID*

2.4.9.2 Dependency on State

This action is allowed in all transport states except in state “*NO_MEDIA_PRESENT*”.

2.4.9.3 Effect on State

This action changes *TransportState* to "*STOPPED*".

2.4.9.4 Errors

Table 2-38 — Error Codes for *Stop()*

ErrorCode	errorDescription	Description
400-499	TBD	See UPnP Device Architecture clause on Control.
500-599	TBD	See UPnP Device Architecture clause on Control.
600-699	TBD	See UPnP Device Architecture clause on Control.
701	Transition not available	The immediate transition from current transport state to desired transport state is not supported by this device.
705	Transport is locked	The transport is *hold locked*. (Some portable mobile devices have a small mechanical toggle switch called a *hold lock switch*. While this switch is ON (the transport is hold locked), the device is guarded against operations such as accidental power on when not in use, or interruption of play or record from accidental pressing of a front panel button or a GUI button.)
718	Invalid InstanceID	The specified *InstanceID* is invalid for this AVTransport.

2.4.10 *Play()*

This action starts playing the resource of the specified instance, at the specified speed, starting at the current position, according to the current play mode. Playing MUST continue until the resource ends or the transport state is changed via actions *Stop()* or *Pause()*. The device MUST do a *best effort* to match the specified play speed. Actually supported speeds can be retrieved from the AllowedValueList of the *TransportPlaySpeed* state variable in the AVTransport service description.

If no *AVTransportURI* is set, the resource being played is device-dependent.

2.4.10.1 Arguments

Table 2-39 — Arguments for *Play()*

Argument	Direction	relatedStateVariable
InstanceID	*IN*	*A_ARG_TYPE_InstanceID*
Speed	*IN*	*TransportPlaySpeed*

2.4.10.2 Dependency on State

This action is allowed in the "*STOPPED*", "*PLAYING*", and "*PAUSED_PLAYBACK*" transport states. In other states the action MAY also succeed or it MAY fail with error code 701.

2.4.10.3 Effect on State

This action changes *TransportState* to "*PLAYING*" and *TransportPlaySpeed* to the value specified in the *Speed* argument of the *Play()* action. If it would take a noticeable amount of time before a human user would actually see or hear the media playing, the AVTransport is allowed to temporarily go to the "*TRANSITIONING*" state before going to "*PLAYING*". This might be appropriate, for example, for devices that need to start buffering or completely download the media before playback can start.

2.4.10.4 Errors

Table 2-40 — Error Codes for *Play()*

errorCode	errorDescription	Description
400-499	TBD	See UPnP Device Architecture clause on Control.
500-599	TBD	See UPnP Device Architecture clause on Control.
600-699	TBD	See UPnP Device Architecture clause on Control.
701	Transition not available	The immediate transition from current transport state to desired transport state is not supported by this device.
702	No contents	The media does not contain any contents that can be played.
703	Read error	The media cannot be read (for example, because of dust or a scratch).
704	Format not supported for playback	The storage format of the currently loaded media is not supported for playback by this device.
705	Transport is locked	The transport is *hold locked*. (Some portable mobile devices have a small mechanical toggle switch called a *hold lock switch*. While this switch is ON (the transport is hold locked), the device is guarded against operations such as accidental power on when not in use, or interruption of play or record from accidental pressing of a front panel button or a GUI button.)
714	Illegal MIME-type	The resource to be played has a MIME-type which is not supported by the AVTransport service.
715	Content 'BUSY'	This indicates that the resource is already in use at this time.
716	Resource not found	The resource to be played cannot be found in the network.
717	Play speed not supported	The specified playback speed is not supported by the AVTransport service.
718	Invalid InstanceID	The specified *InstanceID* is invalid for this AVTransport.
719	DRM error	The action failed because an unspecified DRM error occurred.
720	Expired content	The action failed because the content use validity interval has expired.
721	Non-allowed use	The action failed because the requested content use is disallowed.
722	Can't determine allowed uses	The action failed because the allowed content uses cannot be verified.
723	Exhausted allowed use	The action failed because the number of times this content has been used as requested has reached the maximum allowed number of uses.
724	Device authentication failure	The action failed because of a device authentication failure between the media source device and the media sink device.
725	Device revocation	The action failed because either the media source device or the media sink device has been revoked.

2.4.11 *Pause()*

While the device is in a playing state, that is: *TransportState* is "*PLAYING*", this action halts the progression of the resource that is associated with the specified *InstanceID*. Any visual representation of the resource SHOULD remain displayed in a static manner (for example, the last frame of video remains displayed). Any audio representation of the resource SHOULD be muted. The difference between *Pause()* and *Stop()* is that *Pause()* MUST remain at the current position within the resource and the current resource MUST persist as described above (for example, the current video resource continues to be transmitted/displayed).

When the device is recording, that is: the *TransportState* is "*RECORDING*", the device MUST maintain its current recording position, but does not accept any more data to record. Any data received after the *Pause()* action and before the next *Record()* action will be lost.

2.4.11.1 Arguments

Table 2-41 — Arguments for *Pause()*

Argument	Direction	relatedStateVariable
InstanceID	*IN*	*A_ARG_TYPE_InstanceID*

2.4.11.2 Dependency on State

This action is always allowed while playing or recording. In other cases, the action MAY fail with error code 701.

2.4.11.3 Effect on State

When recording, this action changes *TransportState* to "*PAUSED_RECORDING*". When playing, this action changes *TransportState* to "*PAUSED_PLAYBACK*". The *Pause()* action does not operate as a toggle.

2.4.11.4 Errors

Table 2-42 — Error Codes for *Pause()*

ErrorCode	errorDescription	Description
400-499	TBD	See UPnP Device Architecture clause on Control.
500-599	TBD	See UPnP Device Architecture clause on Control.
600-699	TBD	See UPnP Device Architecture clause on Control.
701	Transition not available	The immediate transition from current transport state to desired transport state is not supported by this device.
705	Transport is locked	The transport is *hold locked*. (Some portable mobile devices have a small mechanical toggle switch called a *hold lock switch*. While this switch is ON (the transport is hold locked), the device is guarded against operations such as accidental power on when not in use, or interruption of play or record from accidental pressing of a front panel button or a GUI button.)
718	Invalid InstanceID	The specified *InstanceID* is invalid for this AVTransport.

2.4.12 *Record()*

This action starts recording on the specified transport instance, at the current position on the media, according to the currently specified recording quality, and returns immediately. If *AVTransportURI* is set (differs from the empty string) then that resource will be recorded. If no *AVTransportURI* is set (equals the empty string), then the source of the content being recorded is device-dependent. In both cases, whether the device outputs the resource to a screen or speakers while recording is device-dependent. If the device implementing the *Record()* action also has a ContentDirectory service, then recorded content will be added to this ContentDirectory in a device-dependent way. Specifically, there is no UPnP mechanism to specify the location of the recorded content in the ContentDirectory hierarchy.

2.4.12.1 Arguments

Table 2-43 — Arguments for *Record()*

Argument	Direction	relatedStateVariable
InstanceID	*IN*	*A_ARG_TYPE_InstanceID*

2.4.12.2 Dependency on State

This action is allowed in the "*STOPPED*" or "*PAUSED_RECORDING*" transport states. In other states the action MAY fail with error code 701.

2.4.12.3 Effect on State

This action changes *TransportState* to "*RECORDING*".

2.4.12.4 Errors

Table 2-44 — Error Codes for *Record()*

errorCode	errorDescription	Description
400-499	TBD	See UPnP Device Architecture clause on Control.
500-599	TBD	See UPnP Device Architecture clause on Control.
600-699	TBD	See UPnP Device Architecture clause on Control.
701	Transition not available	The immediate transition from current transport state to desired transport state is not supported by this device.
705	Transport is locked	The transport is *hold locked*. (Some portable mobile devices have a small mechanical toggle switch called a *hold lock switch*. While this switch is ON (the transport is hold locked), the device is guarded against operations such as accidental power on when not in use, or interruption of play or record from accidental pressing of a front panel button or a GUI button.)
706	Write error	The media cannot be written. (for example, because of dust or a scratch)
707	Media is protected or not writable	The media is write-protected or is of a not writable type.
708	Format not supported for recording	The storage format of the currently loaded media is not supported for recording by this device.
709	Media is full	There is no free space left on the loaded media.
718	Invalid InstanceID	The specified *InstanceID* is invalid for this AVTransport.

2.4.13 *Seek()*

This action starts seeking through the resource controlled by the specified instance - as fast as possible - to the position, specified in the *Target* argument. The value in the *Unit* argument indicates how the *Target* argument needs to be interpreted.

Unit value *"TRACK_NR"* indicates seeking to the beginning of a particular track number. For track-unaware media (such as VCRs), *Seek(InstanceID, "TRACK_NR", "1")* is equivalent to the common *FastReverse* VCR functionality. Special track number *"0"* is used to indicate the end of the media. Hence, *Seek(InstanceID, "TRACK_NR", "0")* is equivalent to the common *FastForward* VCR functionality.

For *Unit* values *"ABS_TIME"*, *"REL_TIME"*, *"ABS_COUNT"*, and *"REL_COUNT"*, the semantics defined by the corresponding state variables MUST be respected. After the *Seek()* action completes, the appropriate state variable must contain the value, specified in the *Target* argument. For example, if the *RelativeTimePosition* state variable contains the value "00:05:30" before the *Seek()* action, then *Seek(InstanceID, "REL_TIME", "00:00:10")* will move the current position to 10 seconds from the beginning of the track and the *RelativeTimePosition* state variable will contain the value "00:00:10" after the *Seek()* action is completed.

For *Unit* value *"REL_FRAME"*, the semantics of the *Target* argument is defined as follows:

- For track-aware media, the *Target* argument contains the desired position in the current track, in terms of frames, measured from the beginning of the current track. The range for the *Target* argument is from "0" to the duration of the current track, measured in number of frames. For track-aware media, the *Target* argument MUST always contain a positive value.

- For track-unaware media (e.g. a single tape), the *Target* argument contains the desired position, in terms of frames, measured from a *zero reference point* on the media. The range for the *Target* argument is from the beginning of the media, measured from the zero reference point to the end of the media, also measured from the zero reference point. For track-unaware media, the *Target* argument can be negative. Indeed, when the zero reference point does not coincide with the beginning of the media, all positions before the zero reference point are expressed as negative values.

For *Unit* value *"FRAME"*, the *Target* argument contains the desired position, in terms of frames, measured from the beginning of the media. The range for the *Target* argument is from "0" to the total duration of the current media, expressed in frames. The *Target* argument MUST always contain a positive value.

The Unit values *"TAPE-INDEX"* and *"REL_TAPE-INDEX"* only apply for track-unaware media. It is assumed that the media contains a set of subsequent 'marks' that indicate some relevant position on the media (a scene change in a video, for instance). The position of these marks and how these marks are inserted on the media is completely device dependent. However, it is further assumed that these marks are sequentially numbered from one to the total number of marks on the media. Furthermore, the first mark is always assumed to be present at the beginning of the media and the last mark is always assumed to be present at the end of the media.

For *Unit* value *"REL_TAPE-INDEX"*, the *Target* argument contains the desired position, in terms of tape index marks, measured from the current position on the media. The range for the *Target* argument is the **i4** data type range. If a value is specified that is outside the range of available tape index marks, then the resulting position will be either the first tape mark position (*Target* < 0) i.e. the beginning of the media, or the last tape mark position (*Target* > 0) i.e. the end of the media.

For *Unit* value *"TAPE-INDEX"*, the *Target* argument contains the desired position, in terms of tape index marks, measured from the beginning of the media. The range for the *Target* argument is from "1" (the first mark on the tape, at the beginning of the tape) to the total number of tape marks on the media. The *Target* argument MUST always contain a positive value.

2.4.13.1 Arguments

Table 2-45 — Arguments for *Seek()*

Argument	Direction	relatedStateVariable
InstanceID	*IN*	*A_ARG_TYPE_InstanceID*
Unit	*IN*	*A_ARG_TYPE_SeekMode*
Target	*IN*	*A_ARG_TYPE_SeekTarget*

2.4.13.2 Dependency on State

This action is allowed in the "*STOPPED*" and "*PLAYING*" transport states, in other states the action MAY fail with error code 701.

2.4.13.3 Effect on State

This action changes *TransportState* to "*TRANSITIONING*" and then returns immediately. When the desired position is reached, *TransportState* will return to the previous transport state (typically "*STOPPED*" or "*PLAYING*"). Note that the new transport state can be detected through the eventing mechanism.

2.4.13.4 Errors

Table 2-46 — Error Codes for *Seek()*

errorCode	errorDescription	Description
400-499	TBD	See UPnP Device Architecture clause on Control.
500-599	TBD	See UPnP Device Architecture clause on Control.
600-699	TBD	See UPnP Device Architecture clause on Control.
701	Transition not available	The immediate transition from current transport state to desired transport state is not supported by this device.
705	Transport is locked	The transport is *hold locked*. (Some portable mobile devices have a small mechanical toggle switch called a *hold lock switch*. While this switch is ON (the transport is hold locked), the device is guarded against operations such as accidental power on when not in use, or interruption of play or record from accidental pressing of a front panel button or a GUI button.)
710	Seek mode not supported	The specified seek mode is not supported by the device.
711	Illegal seek target	The specified seek target is not present on the media or is not specified in terms of the seek mode.
718	Invalid InstanceID	The specified *InstanceID* is invalid for this AVTransport.
719	DRM error	The action failed because an unspecified DRM error occurred.
720	Expired content	The action failed because the content use validity interval has expired.
721	Non-allowed use	The action failed because the requested content use is disallowed.
722	Can't determine allowed uses	The action failed because the allowed content uses cannot be verified.
723	Exhausted allowed use	The action failed because the number of times this content has been used as requested has reached the maximum allowed number of uses.
724	Device authentication failure	The action failed because of a device authentication failure between the media source device and the media sink device.
725	Device revocation	The action failed because either the media source device or the media sink device has been revoked.

2.4.14 *Next()*

This is a convenient action to advance to the next track. This action is functionally equivalent to *Seek(*"*TRACK_NR*", "*CurrentTrackNr+1*"*)*. This action does not *cycle back* to the first track.

2.4.14.1 Arguments

Table 2-47 — Arguments for *Next()*

Argument	Direction	relatedStateVariable
InstanceID	*IN*	*A_ARG_TYPE_InstanceID*

2.4.14.2 Dependency on State

This action is allowed in the "*STOPPED*" and "*PLAYING*" transport states, in other states the action MAY succeed or it MAY fail with error code 701.

2.4.14.3 Effect on State

This action changes *TransportState* to "*TRANSITIONING*" and then returns immediately. When the desired position is reached, *TransportState* will return to the previous transport state (typically "*STOPPED*"). Note that this can be detected through the eventing mechanism.

2.4.14.4 Errors

Table 2-48 — Error Codes for *Next()*

errorCode	errorDescription	Description
400-499	TBD	See UPnP Device Architecture clause on Control.
500-599	TBD	See UPnP Device Architecture clause on Control.
600-699	TBD	See UPnP Device Architecture clause on Control.
701	Transition not available	The immediate transition from current transport state to desired transport state is not supported by this device.
705	Transport is locked	The transport is *hold locked*. (Some portable mobile devices have a small mechanical toggle switch called a *hold lock switch*. While this switch is ON (he transport is hold locked) the device is guarded against operations such as accidental power on when not in use, or interruption of play or record from accidental pressing of a front panel button or a GUI button.)
711	Illegal seek target	The specified seek target is not present on the media.
718	Invalid InstanceID	The specified *InstanceID* is invalid for this AVTransport.
719	DRM error	The action failed because an unspecified DRM error occurred.
720	Expired content	The action failed because the content use validity interval has expired.
721	Non-allowed use	The action failed because the requested content use is disallowed.
722	Can't determine allowed uses	The action failed because the allowed content uses cannot be verified.
723	Exhausted allowed use	The action failed because the number of times this content has been used as requested has reached the maximum allowed number of uses.
724	Device authentication failure	The action failed because of a device authentication failure between the media source device and the media sink device.
725	Device revocation	The action failed because either the media source device or the media sink device has been revoked.

2.4.15 *Previous()*

This is a convenient action to advance to the previous track. This action is functionally equivalent to *Seek(*"*TRACK_NR*", "*CurrentTrackNr-1*"*)*. This action does not *cycle back* to the last track.

2.4.15.1 Arguments

Table 2-49 — Arguments for *Previous()*

Argument	Direction	relatedStateVariable
InstanceID	*IN*	*A_ARG_TYPE_InstanceID*

2.4.15.2 Dependency on State

This action is allowed in the "*STOPPED*" and "*PLAYING*" transport states, in other states the action MAY succeed or it MAY fail with error code 701.

2.4.15.3 Effect on State

This action changes *TransportState* to "*TRANSITIONING*" and then returns immediately. When the desired position is reached, *TransportState* will return to the previous transport state (typically "*STOPPED*"). Note that this can be detected through the eventing mechanism.

2.4.15.4 Errors

Table 2-50 — Error Codes for *Previous()*

errorCode	errorDescription	Description
400-499	TBD	See UPnP Device Architecture clause on Control.
500-599	TBD	See UPnP Device Architecture clause on Control.
600-699	TBD	See UPnP Device Architecture clause on Control.
701	Transition not available	The immediate transition from current transport state to desired transport state is not supported by this device.
705	Transport is locked	The transport is *hold locked*. (Some portable mobile devices have a small mechanical toggle switch called a *hold lock switch*. While this switch is ON (he transport is hold locked) the device is guarded against operations such as accidental power on when not in use, or interruption of play or record from accidental pressing of a front panel button or a GUI button.)
711	Illegal seek target	The specified seek target is not present on the media.
718	Invalid InstanceID	The specified *InstanceID* is invalid for this AVTransport.
719	DRM error	The action failed because an unspecified DRM error occurred.
720	Expired content	The action failed because the content use validity interval has expired.
721	Non-allowed use	The action failed because the requested content use is disallowed.
722	Can't determine allowed uses	The action failed because the allowed content uses cannot be verified.
723	Exhausted allowed use	The action failed because the number of times this content has been used as requested has reached the maximum allowed number of uses.
724	Device authentication failure	The action failed because of a device authentication failure between the media source device and the media sink device.
725	Device revocation	The action failed because either the media source device or the media sink device has been revoked.

2.4.16 *SetPlayMode()*

This action sets the play mode of the specified AVTransport instance.

2.4.16.1 Arguments

Table 2-51 — Arguments for *SetPlayMode()*

Argument	Direction	relatedStateVariable
InstanceID	*IN*	*A_ARG_TYPE_InstanceID*
NewPlayMode	*IN*	*CurrentPlayMode*

2.4.16.2 Dependency on State

None.

2.4.16.3 Effect on State

This action sets the play mode of the specified instance to the specified value. A *subsequent Play()* action for this instance will behave according to the set play mode.

2.4.16.4 Errors

Table 2-52 — Error Codes for *SetPlayMode()*

ErrorCode	errorDescription	Description
400-499	TBD	See UPnP Device Architecture clause on Control.
500-599	TBD	See UPnP Device Architecture clause on Control.
600-699	TBD	See UPnP Device Architecture clause on Control.
712	Play mode not supported	The specified play mode is not supported by the device.
705	Transport is locked	The transport is *hold locked*. (Some portable mobile devices have a small mechanical toggle switch called a *hold lock switch*. While this switch is ON (he transport is hold locked) the device is guarded against operations such as accidental power on when not in use, or interruption of play or record from accidental pressing of a front panel button or a GUI button.)
718	Invalid InstanceID	The specified *InstanceID* is invalid for this AVTransport.

2.4.17 *SetRecordQualityMode()*

This action sets the record quality mode of the specified AVTransport instance.

2.4.17.1 Arguments

Table 2-53 — Arguments for *SetRecordQualityMode()*

Argument	Direction	relatedStateVariable
InstanceID	*IN*	*A_ARG_TYPE_InstanceID*
NewRecordQualityMode	*IN*	*CurrentRecordQualityMode*

2.4.17.2 Dependency on State

None.

2.4.17.3 Effect on State

This action sets *CurrentRecordQualityMode* of the specified instance to the specified record quality mode. A *subsequent* *Record()* action will behave according to the specified record quality mode. This action does not change any ongoing recordings.

2.4.17.4 Errors

Table 2-54 — Error Codes for *SetRecordQualityMode()*

ErrorCode	errorDescription	Description
400-499	TBD	See UPnP Device Architecture clause on Control.
500-599	TBD	See UPnP Device Architecture clause on Control.
600-699	TBD	See UPnP Device Architecture clause on Control.
713	Record quality not supported	The specified record quality is not supported by the device.
718	Invalid InstanceID	The specified *InstanceID* is invalid for this AVTransport.

2.4.18 *GetCurrentTransportActions()*

This action returns the *CurrentTransportActions* state variable for the specified instance.

2.4.18.1 Arguments

Table 2-55 — Arguments for *GetCurrentTransportActions()*

Argument	Direction	relatedStateVariable
InstanceID	*IN*	*A_ARG_TYPE_InstanceID*
Actions	*OUT*	*CurrentTransportActions*

2.4.18.2 Dependency on State

None.

2.4.18.3 Effect on State

None.

2.4.18.4 Errors

Table 2-56 — Error Codes for *GetCurrentTransportActions()*

errorCode	errorDescription	Description
400-499	TBD	See UPnP Device Architecture clause on Control.
500-599	TBD	See UPnP Device Architecture clause on Control.
600-699	TBD	See UPnP Device Architecture clause on Control.
718	Invalid InstanceID	The specified *InstanceID* is invalid for this AVTransport.

2.4.19 *GetDRMState()*

This action returns information associated with the current DRM state of the specified instance. It has no effect on state. The *GetDRMState()* action is optional, but it is required to be implemented when the optional *DRMState* state variable is implemented.

2.4.19.1 Arguments

Table 2-57 — Arguments for *GetDRMState()*

Argument	Direction	relatedStateVariable
InstanceID	*IN*	*A_ARG_TYPE_InstanceID*
CurrentDRMState	*OUT*	*DRMState*

2.4.19.2 Dependency on State

None.

2.4.19.3 Effect on State

None.

2.4.19.4 Errors

Table 2-58 — Error Codes for *GetDRMState()*

errorCode	errorDescription	Description
400-499	TBD	See UPnP Device Architecture clause on Control.
500-599	TBD	See UPnP Device Architecture clause on Control.
600-699	TBD	See UPnP Device Architecture clause on Control.
718	Invalid InstanceID	The specified *InstanceID* is invalid for this AVTransport.

2.4.20 GetStateVariables()

This action returns the current collection of AVTransport state variable names and their respective values that are associated with the AVTransport 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* 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. AVTransport 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.

2.4.20.1 Arguments

Table 2-59 — Arguments for *GetStateVariables()*

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

2.4.20.2 Dependency on State

None.

2.4.20.3 Effect on State

None.

2.4.20.4 Errors

Table 2-60 — Error Codes for *GetStateVariables()*

errorCode	errorDescription	Description
400-499	TBD	See UPnP Device Architecture clause on Control.
500-599	TBD	See UPnP Device Architecture clause on Control.
600-699	TBD	See UPnP Device Architecture clause on Control.
718	Invalid InstanceID	The specified *InstanceID* is invalid for this AVTransport.
726	Invalid StateVariableList	Some of the variables are invalid.
727	Ill-formed CSV List	The CSV list is not well formed.

2.4.21 SetStateVariables()

This action extracts the values from the *StateVariableValuePairs* IN argument and copies these values to the corresponding AVTransport state variables associated with the AVTransport instance indicated by the input argument *InstanceID*. The *AVTransportUDN*, *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 *AVTransportUDN*, *ServiceType* and *ServiceId* input arguments match those of the device. If this is the case, all state variable values will be replaced. Otherwise, the device only sets the state variable values that are relevant. The *StateVariableList* argument is a CSV list of state variable names that were accepted by the AVTransport service. AVTransport service implementations that want to participate in scenarios that use bookmarks MUST implement this optional action.

2.4.21.1 Arguments

Table 2-61 — Arguments for *SetStateVariables()*

Argument	Direction	relatedStateVariable
InstanceID	*IN*	*A_ARG_TYPE_InstanceID*
AVTransportUDN	*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*

2.4.21.2 Dependency on State

None.

2.4.21.3 Effect on State

None.

2.4.21.4 Errors

Table 2-62 — Error Codes for *SetStateVariables()*

errorCode	errorDescription	Description
400-499	TBD	See UPnP Device Architecture clause on Control.
500-599	TBD	See UPnP Device Architecture clause on Control.
600-699	TBD	See UPnP Device Architecture clause on Control.
718	Invalid InstanceID	The specified *InstanceID* is invalid for this AVTransport.
728	Invalid State Variable Value	One of the StateVariableValuePairs contains an invalid value.
729	Invalid Service Type	The specified *ServiceType* is invalid.
730	Invalid Service Id	The specified *ServiceId* is invalid.

2.4.22 Common Error Codes

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

Table 2-63 — Common Error Codes

errorCode	errorDescription	Description
400-499	TBD	See UPnP Device Architecture clause on Control.
500-599	TBD	See UPnP Device Architecture clause on Control.
600-699	TBD	See UPnP Device Architecture clause on Control.
701	Transition not available	The immediate transition from current transport state to desired transport state is not supported by this device.
702	No contents	The media does not contain any contents that can be played.
703	Read error	The media cannot be read (for example, because of dust or a scratch).
704	Format not supported for playback	The storage format of the currently loaded media is not supported for playback by this device.
705	Transport is locked	The transport is *hold locked*. (Some portable mobile devices have a small mechanical toggle switch called a *hold lock switch*. While this switch is ON (he transport is hold locked) the device is guarded against operations such as accidental power on when not in use, or interruption of play or record from accidental pressing of a front panel button or a GUI button.)
706	Write error	The media cannot be written. (for example, because of dust or a scratch)
707	Media is protected or not writable	The media is write-protected or is of a not writable type.
708	Format not supported for recording	The storage format of the currently loaded media is not supported for recording by this device.
709	Media is full	There is no free space left on the loaded media.
710	Seek mode not supported	The specified seek mode is not supported by the device.
711	Illegal seek target	The specified seek target is not present on the media or is not specified in terms of the seek mode.
712	Play mode not supported	The specified play mode is not supported by the device.
713	Record quality not supported	The specified record quality is not supported by the device.
714	Illegal MIME-type	The specified resource has a MIME-type which is not supported by the AVTransport service.
715	Content 'BUSY'	This indicates that the resource is already in use at this time.
716	Resource not found	The specified resource cannot be found in the network.
717	Play speed not supported	The specified playback speed is not supported by the AVTransport service.
718	Invalid InstanceID	The specified *InstanceID* is invalid for this AVTransport.
719	DRM error	The action failed because an unspecified DRM error occurred.
720	Expired content	The action failed because the content use validity interval has expired.
721	Non-allowed use	The action failed because the requested content use is disallowed.
722	Can't determine allowed uses	The action failed because the allowed content uses cannot be verified.
723	Exhausted allowed use	The action failed because the number of times this content has been used as requested has reached the maximum allowed number of uses.
724	Device authentication failure	The action failed because of a device authentication failure between the media source device and the media sink device.

errorCode	errorDescription	Description
725	Device revocation	The action failed because either the media source device or the media sink device has been revoked.
726	Invalid StateVariableList	Some of the variables are invalid.
727	Ill-formed CSV List	The CSV list is not well formed.
728	Invalid State Variable Value	One of the *StateVariableValuePairs* contains an invalid value.
729	Invalid Service Type	The specified *ServiceType* is invalid.
730	Invalid Service Id	The specified *ServiceId* is invalid.

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 clause on Control for more details.

2.5 Theory of Operation

2.5.1 TransportState Control

The main functionality of this service is control over the *TransportState* variable. A state machine depicting the relations between AVTransport actions and *TransportState* values is shown below. In case of any contradictions with the text in the descriptions of the individual actions, the text MUST be considered normative.

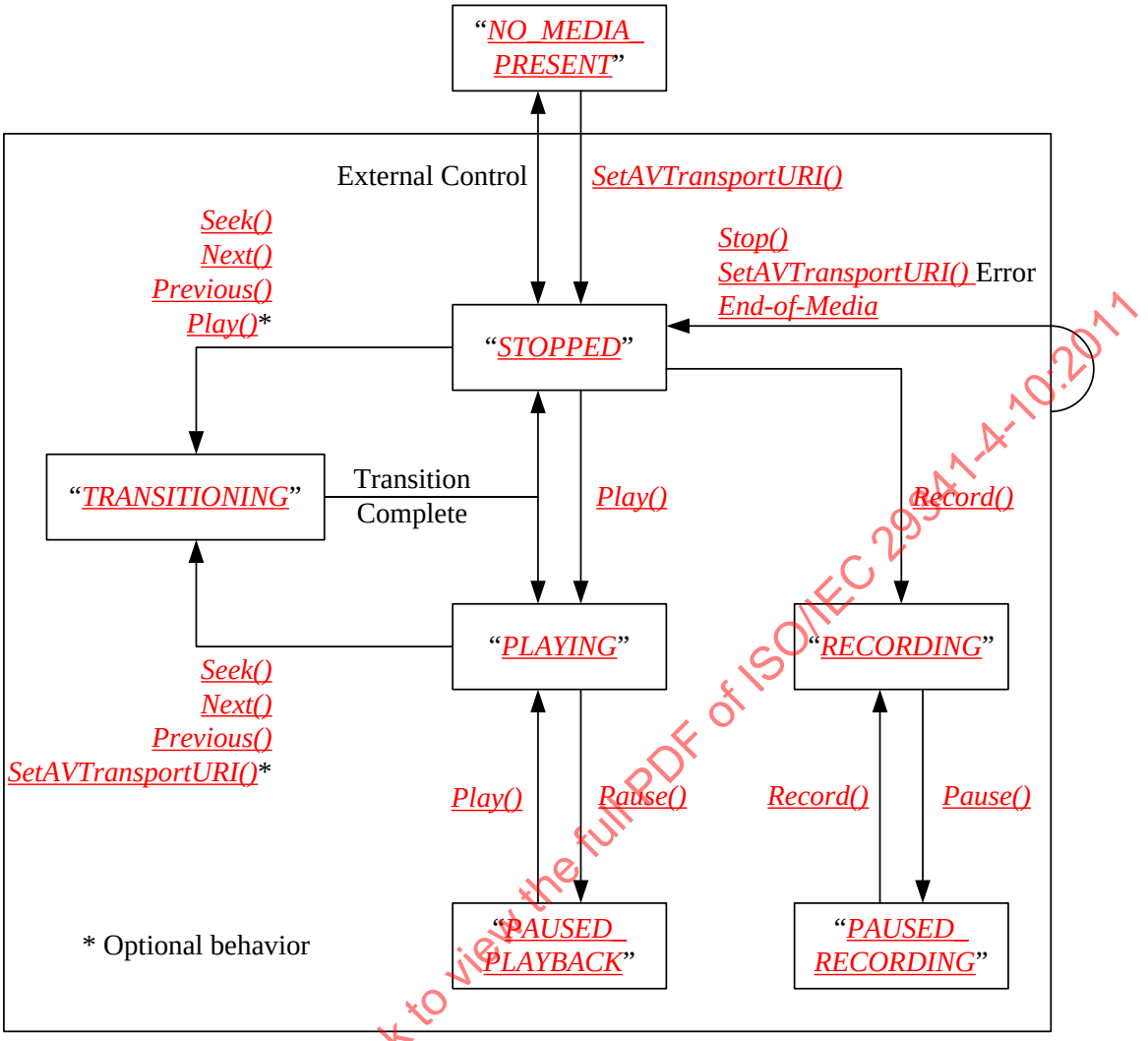

Figure 1: *TransportState* Transitions - INFORMATIVE

Note that the *Stop()* action is allowed in all states except "*NO_MEDIA_PRESENT*", and returns the *TransportState* to the "*STOPPED*" value.

The state machine shows the minimal number of transitions that an AVTransport implementation MUST implement. In addition, any device vendor is allowed to implement more transitions such as, for example, directly from recording to playing mode. For example, nothing prevents a control point from giving a play command during recording, but the AVTransport service simply doesn't require this transition to work, and *is allowed to* return error code 701 (Transition not available). Hence, in such cases, the action might succeed or might not succeed, and a control point should only attempt the action if it has specific knowledge of that vendor's implementation.

In addition, a device vendor is allowed to extend the state diagram above by adding vendor-defined transport states. However, since the semantic meaning of these transport states is not specified, control points that find a AVTransport service in a transport state that they do not understand should refrain from interacting with that AVTransport service (for example, forcing the service into the "*STOPPED*" state). Rather, they should wait until the service transits back into a transport state that they understand.

Besides restrictions on state transitions that are inherent to the device, there might also be *additional* restrictions depending on the *content* whose playback or recording is being controlled. For example, a live stream coming from a broadcast tuner or Internet Radio station cannot be paused. To assist control points that want to reflect these restrictions in

their user interface, an action is defined to return the currently available transport-changing actions – *GetCurrentTransportActions()* – as a comma-separated list of action names. If a control point invokes a transport state changing action that is not in the list returned by *GetCurrentTransportActions()*, a device will return error code 701 (Transition not available).

2.5.2 Transport Settings

Besides control over the transport state, the AVTransport also allows control over various settings related to playback and control. These settings, such as play mode, record mode and record quality mode only take effect on subsequent *Play()* or *Record()* actions. In other words, they do not change the behavior on any ongoing playback or record *session*.

2.5.3 Navigation

The AVTransport allows two types of navigation through the media:

- navigation while producing audio and/or video: this is called *playing*.
- navigation while muting audio and/or video: this is called *seeking*.

Both types of navigation are common in the AV domain, both for audio only and for audio/video media.

Playing is allowed at various speeds. A device is REQUIRED to implement normal speed (x1). Other speeds including *negative speeds* (reverse direction) are OPTIONAL.

The *Seek()* action is very generic and allows a control point to specify a seek operation in various *dimensions*. For example, a control point may instruct a device to position itself 30 seconds from the current position (using the "*RELATIVE_TIME*" seek mode) or to go to track number 12 (using the "*TRACK_NR*" seek mode).

A device does not need to implement all seek modes described in this template (actual supported modes can be retrieved from the allowed value list of variable *A_ARG_TYPE_SeekMode*).

Bookmarking functionality could be implemented by a control point depending on the seek modes supported by the device. Action *GetPositionInfo()* can be used to capture a position on the media, which can be revisited later through action *Seek()*.

2.5.4 AVTransportURI Concept

Uniform Resource Identifiers (URIs) are the Internet standard for resource identification. URIs are simply character strings which identify abstract or physical resources. The complete URI definition is available in the text Uniform Resource Identifiers (URI): Generic Syntax, RFC 2396 (http://www.ietf.org/rfc/rfc2396.txt). URIs MUST be escaped according to the requirements of RFC1738 (http://www.ietf.org/rfc/rfc1738.txt). RFC 1738 requires that unsafe characters be escaped in a valid URI. Characters "<", ">", "\"", "#" and "%" are defined unsafe, because they usually have special meanings, and "{", "}","|","\","^","~","[", "]", and "`" are defined unsafe, because they sometimes get corrupted in mail etc.

Every resource that can be played or recorded via an AVTransport will be modeled in the URI syntax. These resources may be *atomic* resources such as a file containing a song, or may be *non-atomic* or *collection* resources. An example of the latter is an audio playlist, an audio CD or the channel-list in a tuner.

Identifying all resources as URIs provides many advantages. First, using URIs for resource identification follows the existing Internet standard. Second, using URIs for all resources (regardless of transport medium, transport scheme, or content-type) unifies the handling for all types of resources which allows for cleaner, more obvious APIs. Furthermore, using URIs provides extensibility for any transport media or content types. Finally, appending query

strings to URIs allows the ability to pass variables into a dynamically created resource. The basic format is as follows:

```
[scheme]://[host]:[port]/[path]?[query string]
```

The table below lists how the *protocolInfo* definitions from the ConnectionManager specification relate to valid URLs. The annex provides a more detailed explanation per protocol.

Table 2-64 — Allowed AVTransportURIs

ProtocolInfo	URI Scheme	Reference clause
http-get	http	A.1
rtsp-rtp-udp	rtsp	A.2
internal	file	A.3
iec61883	*Vendor-defined*	A.4
registered ICANN domain name of vendor	*Vendor-defined*	A.5

2.5.5 AVTransport Abstraction

Via the *SetAVTransportURI()* action an AVTransport service instance is *bound* to a content resource. Content resources are exposed by the ContentDirectory service. A content resource can represent a single atomic piece of content (for example, a single song), or a collection of contents (for example, a CD disc or playlist). The *types* of content resources that can be sent or received by a device are exposed by the *GetProtocolInfo()* action of the device's ConnectionManager service.

Once a content resource is bound to an AVTransport instance, the instance maps the resource to a flat sequence of tracks. This sequence can then be navigated via actions *Seek()*, *Next()* and *Previous()*. For example, a resource pointing to a single audio song is mapped to 1 track, while a resource pointing to some audio playlist format is mapped to a sequence of tracks where each playlist entry maps to a single track. In case of embedded playlists, entries are mapped to tracks using a *depth-first* traversal. Playlist entries that cannot be handled by the AVTransport (for example, unknown audio formats, etc.), MUST be skipped by the AVTransport. Whether those entries are eliminated immediately (not included in the number of tracks) or immediately before playback, is device-dependent.

In addition, an AVTransport might provide other means of navigation, such as time-based seeking.

The AVTransport abstracts and minimizes the differences between various specific transport media such as tapes, discs and solid-state media. The tables below gives an overview on how generic AVTransport concepts such as track and *Next()* and *Previous()* actions apply to certain specific types of *AVTransportURIs*. The precise mapping is implementation-dependent.

Table 2-65 — Example mappings of resources type to track sequences

AVTransport URI	Track concept	Number of Tracks	Current track duration	*Next()*/*Previous()* actions (in normal PlayMode)
Audio CD	1 track	all tracks on the CD	duration of track	Next or previous track on the CD.
Audio CD Changer	1 track	all tracks of all CDs in the CD changer combined	duration of track	Next or previous track on the CD, also transition to previous or next CD in the changer if current track is the first or last one on the current disc.
Audio Playlist (HDD/SolidState-based player)	1 entry in a playlist	all entries of the playlist, including entries of embedded playlists	duration of the playlist entry	Next or previous entry in the playlist; in case of embedded playlists, navigate using a depth-first traversal.
Video DVD-Volume	1 chapter	all chapters on the DVD-Volume	duration of chapter	Next or previous chapter on the DVD.
Video DVD Changer	1 volume	all volumes of all DVD discs in the DVD changer combined	duration of volume	Next or previous volume on the DVD, also transition to previous or next DVD in the changer if current volume is the first or last one on the current disc.
VCR (Tape)	all content on the tape	1	tape-length, or 0 if the tape-length is unknown	No effect.
List-based Tuner	1 video channel or 1 radio station	all video channels or radio stations of the tuner channel list	0	Next or previous video channel or radio station in the list.; in case of major channels containing minor channels, use a depth-first traversal.
Frequency-based Tuner	1 frequency	number of selectable frequencies	0	Increment or decrement frequency by device-dependent amount.
PVR – Tuner subsystem	1 video channel	all live video channels of the PVR	0	Next of previous video channel in the list. In case of major channels containing minor channels, use a depth-first traversal.
PVR –Collection of Stored programs	1 program	all programs of the PVR-store	duration of the program	Next or previous program in the collection.
PVR – Single Stored program	1 program	1	duration of the program	No effect.
EPF	1 image	all files of the slide show	display time of the slide in the slide show	Next of previous slide in a slide show.

The type of resource (audio, video, image, etc.) and storage media typically affect the way the resource can be searched (seek modes), trick/play modes, and whether pausing is possible. The table below gives examples for a number of resource types.

Table 2-66 — Example seek modes, play modes and transport actions, per resource type

AVTransport URI	Applicable (not required) Seek modes	Applicable (not required) Play modes	Pausing possible
Audio CD	TRACK_NR	NORMAL, SHUFFLE, REPEAT_ONE, REPEAT_ALL, RANDOM, DIRECT_1, INTRO	yes
Audio CD Changer	TRACK_NR	NORMAL, SHUFFLE, REPEAT_ONE, REPEAT_ALL, RANDOM, DIRECT_1, INTRO	yes
Audio Playlist (HDD/SolidState-based player)	TRACK_NR	NORMAL, SHUFFLE, REPEAT_ONE, REPEAT_ALL, RANDOM, DIRECT_1, INTRO	yes
Video DVD-Volume	TRACK_NR, FRAME	NORMAL	yes
Video DVD Changer	TRACK_NR, FRAME	NORMAL	yes
VCR (Tape)	TRACK_NR, ABS_TIME, REL_TIME, ABS_COUNT, REL_COUNT, TAPE_INDEX, FRAME	NORMAL	yes
List-based Tuner	TRACK_NR	NORMAL	no
Frequency-based Tuner	TRACK_NR, CHANNEL_FREQ	NORMAL	no
PVR – Tuner subsystem	TRACK_NR, ABS_TIME, REL_TIME, FRAME	NORMAL	yes
PVR – Collection of Stored programs	TRACK_NR, ABS_TIME, REL_TIME, FRAME	NORMAL	yes
PVR – Stored program	TRACK_NR, ABS_TIME, REL_TIME, FRAME	NORMAL	yes
EPF	TRACK_NR, ABS_TIME	NORMAL, SHUFFLE, REPEAT_ONE, REPEAT_ALL, RANDOM, DIRECT_1	yes

2.5.6 Supporting Multiple Virtual Transports

The UPnP Architecture v1.0 requires the number of service instances in a device to be static. In certain cases it is desirable for devices to offer a dynamic number of *virtual* service instances. A control point needs to be able to control and receive events from each virtual instance individually.

In the AVTransport service case, some devices will be able to serve content to a number of clients simultaneously. For these MediaServer devices, the actual number of clients, typically renderer or recording/dubbing devices, may be fairly large, and not statically known. This service is applicable for these types of devices, as well as traditional – more static – types of devices.

A generic strategy to achieve this is as follows:

- Have a single static UPnP service instance in a device (hence, a single UPnP **serviceId**).
  - In the AVTransport case, a MediaServer device or MediaRenderer device can have a single UPnP AVTransport instance.
- Define the notion of a *virtual instance identifier* (this is *not* the UPnP **serviceId**).
  - In the AVTransport case, a **ui4** value.
- Add to all actions of the service definition an input argument that holds the virtual instance identifier to which the action applies.

    - All actions in the AVTransport service, such as *Play()*, *Stop()*, *Pause()*, have as first input argument of type **ui4** that identifies the instance (*InstanceID*).
- Add an evented state variable (in this case *LastChange*) to the service that holds both the instance identifier and the name and value of the latest state change of this instance. All other variables are not directly evented.
- Define a *factory* method, in the same service or in a related service that a control point can call to obtain a new instance identifier. This factory method MUST return an error when no instances are available anymore.
    - For AVTransport, *ConnectionManager::PrepareForConnection()* serves as the factory method for obtaining a new *InstanceID* for AVTransport. In case the factory method is not present, reserved *InstanceID* value 0 can be used.
- OPTIONALLY, define a *cleanup* method via which a control point can indicate that the obtained instance identifier will no longer be used, and can be reused for allocation to other control points. To accommodate the situation where a control point leaves the UPnP network before calling the *cleanup* action, there needs to be an automatic cleanup mechanism implemented by the device as well. This mechanism will be device- or maybe even vendor-specific, and needs to be described in the device template.
    - For AVTransport, *ConnectionManager::ConnectionComplete()* serves as the factory method for releasing an *InstanceID*.
- OPTIONALLY, define an action (and associated state variable) to retrieve the list of *currently active/allocated* instance identifiers. This is useful for control points that enter a new network and want to discover what services are currently available. It also enables control points to manually *cleanup* the whole network.
    - For AVTransport, *ConnectionManager::GetCurrentConnectionIDs()* and *ConnectionManager::GetCurrentConnectionInfo()* are used for this purpose.

The *factory* action, the *cleanup* action and the action to retrieve the *currently active/allocated* instances SHOULD normally be grouped in the same service.

To make a service that has been parameterized by instance identifiers also usable in a context (device) where no factory and cleanup actions are applicable, one or more fixed instance identifier values can be defined that a control point can directly pass in to the service actions. In the AVTransport case, special instance identifier value 0 has been defined for this purpose. The device template using such a static mechanism MUST describe the semantics of that single virtual instance. For example, a vendor-specific device type that does not implement a MediaServer device but only an AVTransport service can be controlled by a control point via, for example, *AVTransport::Play(0, 1)*, *AVTransport::Stop(0)*, etc.

2.5.7 Playlist Playback

An important use of this service will be to control playback of an (audio) playlist. In this case, the URI of the playlist file MUST be bound to the AVTransport instance via *SetAVTransportURI()*. The playlist itself only needs to be processed by the AVTransport implementation on the renderer device, the control point itself does not need to understand or parse the playlist file. Song after song of the playlist can be played without any operation required by the control point. For example, the control point may power down, control some other devices or leave the house, without affecting the playlist playback.

When a control point has a display and wants to show meta-data of the currently playing resource it can:

- subscribe to AVTransport events so it always knows the current transport state and track information
- use *CurrentTrackURI* to obtain meta-data of the currently playing track via the *ContentDirectory::Search()* action of the ContentDirectory service

- use *CurrentTrackMetaData* to obtain any meta-data of the currently playing track from the AVTransport service directly

Whether an AVTransport implementation can deal with relative URLs that may be present inside a playlist file is device-dependent.

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>SetAVTransportURI</name>
         <argumentList>
            <argument>
               <name>InstanceID</name>
               <direction>in</direction>
               <relatedStateVariable>
                  A_ARG_TYPE_InstanceID
               </relatedStateVariable>
            </argument>
            <argument>
               <name>CurrentURI</name>
               <direction>in</direction>
               <relatedStateVariable>
                  AVTransportURI
               </relatedStateVariable>
            </argument>
            <argument>
               <name>CurrentURIMetaData</name>
               <direction>in</direction>
               <relatedStateVariable>
                  AVTransportURIMetaData
               </relatedStateVariable>
            </argument>
         </argumentList>
      </action>
      <action>
         <name>SetNextAVTransportURI</name>
         <argumentList>
            <argument>
               <name>InstanceID</name>
               <direction>in</direction>
               <relatedStateVariable>
                  A_ARG_TYPE_InstanceID
               </relatedStateVariable>
            </argument>
            <argument>
               <name>NextURI</name>
               <direction>in</direction>
               <relatedStateVariable>
                  NextAVTransportURI
               </relatedStateVariable>
            </argument>
            <argument>
               <name>NextURIMetaData</name>
               <direction>in</direction>
               <relatedStateVariable>
                  NextAVTransportURIMetaData
               </relatedStateVariable>
            </argument>
         </argumentList>
      </action>
      <action>
```

```
<name>GetMediaInfo</name>
<argumentList>
   <argument>
      <name>InstanceID</name>
      <direction>in</direction>
      <relatedStateVariable>
         A_ARG_TYPE_InstanceID
      </relatedStateVariable>
   </argument>
   <argument>
      <name>NrTracks</name>
      <direction>out</direction>
      <relatedStateVariable>
         NumberOfTracks
      </relatedStateVariable>
   </argument>
   <argument>
      <name>MediaDuration</name>
      <direction>out</direction>
      <relatedStateVariable>
         CurrentMediaDuration
      </relatedStateVariable>
   </argument>
   <argument>
      <name>CurrentURI</name>
      <direction>out</direction>
      <relatedStateVariable>
         AVTransportURI
      </relatedStateVariable>
   </argument>
   <argument>
      <name>CurrentURIMetaData</name>
      <direction>out</direction>
      <relatedStateVariable>
         AVTransportURIMetaData
      </relatedStateVariable>
   </argument>
   <argument>
      <name>NextURI</name>
      <direction>out</direction>
      <relatedStateVariable>
         NextAVTransportURI
      </relatedStateVariable>
   </argument>
   <argument>
      <name>NextURIMetaData</name>
      <direction>out</direction>
      <relatedStateVariable>
         NextAVTransportURIMetaData
      </relatedStateVariable>
   </argument>
   <argument>
      <name>PlayMedium</name>
      <direction>out</direction>
      <relatedStateVariable>
         PlaybackStorageMedium
      </relatedStateVariable>
   </argument>
   <argument>
      <name>RecordMedium</name>
      <direction>out</direction>
      <relatedStateVariable>
         RecordStorageMedium
      </relatedStateVariable>
   </argument>
   <argument>
      <name>WriteStatus</name>
      <direction>out</direction>
      <relatedStateVariable>
         RecordMediumWriteStatus
```

```
            </relatedStateVariable>
         </argument>
      </argumentList>
   </action>
   <action>
      <name>GetMediaInfo_Ext</name>
      <argumentList>
         <argument>
            <name>InstanceID</name>
            <direction>in</direction>
            <relatedStateVariable>
               A_ARG_TYPE_InstanceID
            </relatedStateVariable>
         </argument>
         <argument>
            <name>CurrentType</name>
            <direction>out</direction>
            <relatedStateVariable>
               CurrentMediaCategory
            </relatedStateVariable>
         </argument>
         <argument>
            <name>NrTracks</name>
            <direction>out</direction>
            <relatedStateVariable>
               NumberOfTracks
            </relatedStateVariable>
         </argument>
         <argument>
            <name>MediaDuration</name>
            <direction>out</direction>
            <relatedStateVariable>
               CurrentMediaDuration
            </relatedStateVariable>
         </argument>
         <argument>
            <name>CurrentURI</name>
            <direction>out</direction>
            <relatedStateVariable>
               AVTransportURI
            </relatedStateVariable>
         </argument>
         <argument>
            <name>CurrentURIMetaData</name>
            <direction>out</direction>
            <relatedStateVariable>
               AVTransportURIMetaData
            </relatedStateVariable>
         </argument>
         <argument>
            <name>NextURI</name>
            <direction>out</direction>
            <relatedStateVariable>
               NextAVTransportURI
            </relatedStateVariable>
         </argument>
         <argument>
            <name>NextURIMetaData</name>
            <direction>out</direction>
            <relatedStateVariable>
               NextAVTransportURIMetaData
            </relatedStateVariable>
         </argument>
         <argument>
            <name>PlayMedium</name>
            <direction>out</direction>
            relatedStateVariable>
               PlaybackStorageMedium
            </relatedStateVariable>
         </argument>
```