ConnectionManager:3 Service

For UPnP Version 1.0
Status: Standardized DCP (SDCP)
Date: March 31, 2013
Service Template Version 3.0

This Standardized DCP has been adopted as a Standardized DCP by the Steering Committee of the UPnP Forum, pursuant to Section 2.1(c)(ii) of the UPnP Forum Membership Agreement. UPnP Forum Members have rights and licenses defined by Section 3 of the UPnP Forum Membership Agreement to use and reproduce the Standardized DCP in UPnP Compliant Devices. All such use is subject to all of the provisions of the UPnP Forum Membership Agreement.

© 2014, UPnP Forum. All rights Reserved.

Authors	Company
Alan Presser	Allegrosoft
Wouter van der Beek	Cisco Systems
Gary Langille	Echostar
Gerrie Shults	HP
Raj Bopardikar	Intel
Nelson Kidd	Intel
John Ritchie	Intel
Mark Walker	Intel
Keith Miller	Intel
Richard Bardini	Intel
Sungjoon Ahn	LG Electronics
Changhyun Kim	LG Electronics
Seung R. Yang (Co-Chair)	LG Electronics
Masatomo Hori	Matsushita Electric (Panasonic)
Matthew Ma	Matsushita Electric (Panasonic)
Jack Unverferth	Microsoft
Keith Miller (Co-Chair)	Nokia
Wouter van der Beek	Philips
Wim Bronnenberg	Philips
Jeffrey Kang	Philips
Geert Knapen	Philips
Russell Berkoff	Pioneer
Irene Shen	Pioneer
Russell Berkoff (Vice-Chair)	Samsung
Richard Bardini	Sony
Norifumi Kikkawa	Sony
Jonathan Tourzan	Sony
Yasuhiro Morioka	Toshiba
Jeffrey Kang	TP Vision
Nicholas Frame	TP Vision
Note: The UPnP Forum in no way guarantees the accuracy or completeness of this author list and in no way implies any rights for or support from those members listed. This list is not the specifications' contributor list that is kept on the UPnP Forum's website.	

CONTENTS

List of Tables

1 Scope

This service definition is compliant with the UPnP Device Architecture version 1.0 [14].

This service-type enables modeling of streaming capabilities of A/V devices, and binding of those capabilities between devices. Each device that is able to send or receive a stream according to the UPnP AV Architecture will have 1 instance of the ConnectionManager service. This service provides a mechanism for control points to:

a) Perform capability matching between source/server devices and sink/renderer devices,

b) Find information about currently ongoing transfers in the network,

c) Setup and teardown connections between devices (when required by the streaming protocol).

The ConnectionManager service is generic enough to properly abstract different kinds of streaming mechanisms, such as HTTP-based streaming, RTSP/RTP-based and 1394-based streaming.

The ConnectionManager enables control points to abstract from physical media interconnect technology when making connections. The term ‘stream’ used in this service template refers to both analog and digital data transfer.

2 Normative references

The following documents, in whole or in part, are normatively referenced in this document and are indispensable for its application. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

[1] – *XML Schema for RenderingControl AllowedTransformSettings*, UPnP Forum, March 31, 2013.
Available at: http://www.upnp.org/schemas/av/AllowedTransformSettings-v1-20130331.xsd.
Latest version available at: http://www.upnp.org/schemas/av/AllowedTransformSettings.xsd.

[2] – *AV Datastructure Template:1*, UPnP Forum, March 31, 2013.
Available at: http://www.upnp.org/specs/av/UPnP-av-AVDataStructureTemplate-v1-20130331.pdf.
Latest version available at: http://www.upnp.org/specs/av/UPnP-av-AVDataStructureTemplate-v1.pdf.

[3] – *XML Schema for UPnP AV Common XML Data Types*, UPnP Forum, March 31, 2013.
Available at: http://www.upnp.org/schemas/av/av-v3-20130331.xsd.
Latest version available at: http://www.upnp.org/schemas/av/av.xsd.

[4] – *XML Schema for UPnP AV Common XML Structures*, UPnP Forum, March 31, 2013.
Available at: http://www.upnp.org/schemas/av/avs-v3-20130331.xsd.
Latest version available at: http://www.upnp.org/schemas/av/avs.xsd.

[5] – *AVTransport:3*, UPnP Forum, March 31, 2013.
Available at: http://www.upnp.org/specs/av/UPnP-av-AVTransport-v3-Service-20130331.pdf.
Latest version available at: http://www.upnp.org/specs/av/UPnP-av-AVTransport-v3-Service.pdf.

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

[7] – *ContentDirectory:4*, UPnP Forum, March 31, 2013.
Available at: http://www.upnp.org/specs/av/UPnP-av-ContentDirectory-v4-Service-20130331.pdf.

Latest version available at: http://www.upnp.org/specs/av/UPnP-av-ContentDirectory-v4-Service.pdf.

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

[9] – *ConnectionManager:3*, UPnP Forum, March 31, 2013.
Available at: http://www.upnp.org/specs/av/UPnP-av-ConnectionManager-v3-Service-20130331.pdf.
Latest version available at: http://www.upnp.org/specs/av/UPnP-av-ConnectionManager-v3-Service.pdf.

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

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

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

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

[14] – *UPnP Device Architecture, version 1.0*, UPnP Forum, October 15, 2008.
Available at: http://www.upnp.org/specs/arch/UPnP-arch-DeviceArchitecture-v1.0-20081015.pdf.
Latest version available at: http://www.upnp.org/specs/arch/UPnP-arch-DeviceArchitecture-v1.0.pdf.

[15] – *XML Schema for ContentDirectory Structure and Metadata (DIDL-Lite)*, UPnP Forum, March 31, 2013.
Available at: http://www.upnp.org/schemas/av/didl-lite-v3-20130331.xsd.
Latest version available at: http://www.upnp.org/schemas/av/didl-lite.xsd.

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

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

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

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

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

[21] – *RenderingControl:3*, UPnP Forum, March 31, 2013.
Available at: http://www.upnp.org/specs/av/UPnP-av-RenderingControl-v3-Service-20130331.pdf.
Latest version available at: http://www.upnp.org/specs/av/UPnP-av-RenderingControl-v3-Service.pdf.

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

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

[24] – *XML Schema for AVTransport PlaylistInfo*, UPnP Forum, March 31, 2013.
Available at: http://www.upnp.org/schemas/av/rpl-v1-20130331.xsd.
Latest version available at: http://www.upnp.org/schemas/av/rpl.xsd.

[25] – *ScheduledRecording:2*, UPnP Forum, March 31, 2013.
Available at: http://www.upnp.org/specs/av/UPnP-av-ScheduledRecording-v2-Service-20130331.pdf.
Latest version available at: http://www.upnp.org/specs/av/UPnP-av-ScheduledRecording-v2-Service.pdf.

[26] – *XML Schema for ScheduledRecording Metadata and Structure*, UPnP Forum, March 31, 2013.
Available at: http://www.upnp.org/schemas/av/srs-v2-20130331.xsd.
Latest version available at: http://www.upnp.org/schemas/av/srs.xsd.

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

[28] – *XML Schema for RenderingControl TransformSettings*, UPnP Forum, March 31, 2013.
Available at: http://www.upnp.org/schemas/av/TransformSettings-v1-20130331.xsd.
Latest version available at: http://www.upnp.org/schemas/av/TransformSettings.xsd.

[29] – *XML Schema for ContentDirectory Metadata*, UPnP Forum, March 31, 2013.
Available at: http://www.upnp.org/schemas/av/upnp-v4-20130331.xsd.
Latest version available at: http://www.upnp.org/schemas/av/upnp.xsd.

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

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

[32] – *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.

[33] – *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.

[34] – *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.

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

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

[37] – *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.

[38] – *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.

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

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

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

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

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

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

[45] – *IEEE-P1733™ (Draft 2.2) – Audio Video Bridge Layer 3 Transport Protocol*, International Institute of Electrical and Electronics Engineers, April 20, 2009.
Available at: http://grouper.ieee.org/groups/1733.

[46] – *IETF RFC 3550, RTP: A Transport Protocol for Real-Time Applications,* H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson, July 2003.
Available at: http://www.ietf.org/rfc/rfc3550.txt.

[47] – *AVArchitecture:2*, UPnP Forum, March 31, 2013.
Available at: http://www.upnp.org/specs/av/UPnP-av-AVArchitecture-v2-20130331.pdf.
Latest version available at: http://www.upnp.org/specs/av/UPnP-av-AVArchitecture-v2.pdf.

3 Terms, definitions, symbols and abbreviations

For the purposes of this document, the terms and definitions given in [14] and the following subclauses 3.1 and 3.2 apply.

3.1 Provisioning terms

3.1.1 allowed

A

The definition or behavior is allowed.

3.1.2
conditionally allowed
CA
The definition or behavior depends on a condition. If the specified condition is met, then the definition or behavior is allowed, otherwise it is not allowed.

3.1.3
conditionally required
CR
The definition or behavior depends on a condition. If the specified condition is met, then the definition or behavior is required. Otherwise the definition or behavior is allowed as default unless specifically defined as not allowed.

3.1.4
required
R
The definition or behavior is required.

3.1.5
R/A
Used in a table column heading to indicate that each abbreviated entry in the column declares the provisioning status of the item named in the entry's row.

3.1.6
X
Vendor-defined, non-standard.

3.1.7
-D
Declares that the item referred to is deprecated, when it is appended to any of the other abbreviated provisioning terms.

3.1.8
CSV list (or CSV)
Comma separated value list. List—or one-dimensional array—of values contained in a string and separated by commas

3.2 Symbols

3.2.1
::
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.

4 Notations and Conventions

4.1 Notation

- UPnP interface names defined in the UPnP Device Architecture specification [14] are styled in **green bold underlined** text.
- UPnP interface names defined outside of the UPnP Device Architecture specification [14] are styled in *red italic underlined* text.
- Some additional non-interface names and terms are styled in *italic* text.
- Words that are emphasized are also styled in *italic* text. The difference between italic terms and italics for emphasis will be apparent by context.
- Strings that are to be taken literally are enclosed in “double quotes”.

4.1.1 Data Types

Data type definitions come from three sources:

- All state variable and action argument data types are defined in [14].
- Basic data types for properties are defined in [34].
- Additional data types for properties are defined in the XML schema(s) (see [3]) associated with this service.

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 shall 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 shall be accepted. Nevertheless, it is strongly recommended that all Boolean properties be represented as "*0*" and "*1*".

4.1.2 Strings Embedded in Other Strings

Some string variables and arguments described in this document contain substrings that shall 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 subclause 4.2.2) 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

4.1.3 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 [19].

4.1.3.1 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 shall 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 in Table 1:

Table 1 — EBNF Operators

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

4.2 Derived Data Types

4.2.1 Summary

Subclause 4.2 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 in subclause 4.2.2 applies to both string data types.

4.2.2 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 [14], 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 (all values can be of different types). Lists may also consist of repeated occurrences of homogeneous or heterogeneous subsequences, all of which have the same syntax and semantics (same number of values, same value types and in the same order). The data type of a homogeneous list is **string** or xsd:string and denoted by CSV (*x*), where *x* is the type of the individual values. The data type of a heterogeneous list is also **string** or xsd:string and denoted by CSV (*x, y, z*), where *x*, *y* and *z* are the types of the individual values. If the number of values in the heterogeneous list is too large to show each type individually, that variable type is represented as CSV (heterogeneous), and the variable description includes additional information as to the expected sequence of values appearing in the list and their corresponding types. The data type of a repeated subsequence list is **string** or xsd:string and denoted by CSV ({a,b,c},{*x*, *y*, *z*}), where a, b, c, *x*, *y* and *z* are the types of the individual values in the subsequence and the subsequences may be repeated zero or more times.

- A list is represented as a **string** type (for state variables and action arguments) or xsd:string type (for properties).
- Commas separate values within a list.
- Integer values are represented in CSVs with the same syntax as the integer data type specified in [14] (that is: allowed leading sign, allowed 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 [14]: **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 [34]: 0, false, 1, true.
- Escaping conventions for the comma and backslash characters are defined in 4.1.2.

- 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 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.

4.3 Management of XML Namespaces in Standardized DCPs

UPnP specifications make extensive use of XML namespaces. This enables 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 shall 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 shall be locally unique to the document. Any valid XML no-colon-name may be used. And, in formal XML usage, different XML documents may use different namespace prefixes to refer to the same namespace. The creation and use of the namespace prefix was standardized by the W3C XML Committee in [32] strictly as a convenient local shorthand replacement for the full URI name of a namespace in individual documents.

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

For the same reason that namespace prefixes are convenient in XML documents, it is convenient in specification text to refer to namespaces using a namespace prefix. Therefore, this specification declares a “standard” prefix for all XML namespaces used herein. In addition, this specification expands the scope where these prefixes have meaning, beyond a single XML document, to all of its text, XML examples, and certain string-valued properties. This expansion of scope *does not* supersede XML rules for usage in documents, it only augments and complements them in important contexts that are out-of-scope for the XML specifications. For example, action arguments which refer to CDS properties, such as the *SearchCriteria* argument of the *Search()* action or the *Filter* argument of the *Browse()* action, shall use the predefined namespace prefixes when referring to CDS properties (“upnp:”, “dc:”, etc).

All of the namespaces used in this specification are listed in Table 3 and Table 4. For each such namespace, Table 3 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 ScheduledRecording service depends on and refers to the ContentDirectory 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 4 to cross-reference additional namespace information. Table 4 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 2 for its associated schema.

The normative definitions for these namespaces are the documents referenced in Table 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 3 — Namespace Definitions

Standard Name-space Prefix	Namespace Name	Namespace Description	Normative Definition Document Reference
AV Working Committee defined namespaces			
atrs	urn:schemas-upnp-org:av:AllowedTransformSettings	*AllowedTransformSettings* and *AllowedDefaultTransformSettings* state variables for RenderingControl	[21]
av	urn:schemas-upnp-org:av:av	Common data types for use in AV schemas	[3]
avdt	urn:schemas-upnp-org:av:avdt	Datastructure Template	[2]
avs	urn:schemas-upnp-org:av:avs	Common structures for use in AV schemas	[4]
avt-event	urn:schemas-upnp-org:metadata-1-0/AVT/	Evented *LastChange* state variable for AVTransport	[5]
cds-event	urn:schemas-upnp-org:av:cds-event	Evented *LastChange* state variable for ContentDirectory	[7]

Standard Name-space Prefix	Namespace Name	Namespace Description	Normative Definition Document Reference
cm-dciu	urn:schemas-upnp-org:av:cm-deviceClockInfoUpdates	Evented *DeviceClockInfoUpdates* state variable for ConnectionManager	[9]
cm-ftrlst	urn:schemas-upnp-org:av:cm-featureList	*FeatureList* state variable for ConnectionManager	[9]
didl-lite	urn:schemas-upnp-org:metadata-1-0/DIDL-Lite/	Structure and metadata for ContentDirectory	[7]
dmo	urn:schemas-upnp.org:av:dmo	Evented *DeviceMode* state variable for ContentDirectory	[7]
dmor	urn:schemas-upnp.org:av:dmor	*A_ARG_TYPE_DeviceModeRequest* state variable for ContentDirectory	[7]
dmos	urn:schemas-upnp.org:av:dmos	*DeviceModeStatus* state variable for ContentDirectory	[7]
pi	urn:schemas-upnp.org:av:pi	*PermissionsInfo* state variable for ContentDirectory	[7]
rcs-event	urn:schemas-upnp-org:metadata-1-0/RCS/	Evented *LastChange* state variable for RenderingControl	[21]
rii	urn:schemas-upnp-org:av:rii	*A_ARG_TYPE_RenderingInfoList* state variable for ConnectionManager	[9]
rpl	urn:schemas-upnp-org:av:rpl	*A_ARG_TYPE_PlaylistInfo* state variable for AVTransport	[5]
srs	urn:schemas-upnp-org:av:srs	Metadata and structure for ScheduledRecording	[25]
srs-event	urn:schemas-upnp-org:av:srs-event	Evented *LastChange* state variable for ScheduledRecording	[25]
trs	urn:schemas-upnp-org:av:TransformSettings	*TransformSettings* and *DefaultTransformSettings* state variables for RenderingControl	[21]
upnp	urn:schemas-upnp-org:metadata-1-0/upnp/	Metadata for ContentDirectory	[7]
Externally defined namespaces			
dc	http://purl.org/dc/elements/1.1/	Dublin Core	[13]
xsd	http://www.w3.org/2001/XMLSchema	XML Schema Language 1.0	[33], [34]
xsi	http://www.w3.org/2001/XMLSchema-instance	XML Schema Instance Document schema	[33] 2.6 & 3.2.7
xml	http://www.w3.org/XML/1998/namespace	The “xml:” Namespace	[30]

Table 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			
atrs	AllowedTransformSettings-v*n-yyyymmdd*.xsd AllowedTransformSettings-v*n*.xsd AllowedTransformSettings.xsd	<TransformList>	[1]
av	av-v*n-yyyymmdd*.xsd av-v*n*.xsd av.xsd	*n/a*	[3]
avdt	avdt-v*n-yyyymmdd*.xsd avdt-v*n*.xsd avdt.xsd	<AVDT>	[2]
avs	avs-v*n-yyyymmdd*.xsd avs-v*n*.xsd avs.xsd	<Capabilities> <Features> <stateVariableValuePairs>	[4]
avt-event	avt-event-v*n-yyyymmdd*.xsd avt-event-v*n*.xsd avt-event.xsd	<Event>	[6]
cds-event	cds-event-v*n-yyyymmdd*.xsd cds-event-v*n*.xsd cds-event.xsd	<StateEvent>	[8]
cm-dciu	cm-deviceClockInfoUpdates-v*n-yyyymmdd*.xsd cm-deviceClockInfoUpdates-v*n*.xsd cm-deviceClockInfoUpdates.xsd	<DeviceClockInfoUpdates>	[10]
cm-ftrlst	cm-featureList-v*n-yyyymmdd*.xsd cm-featureList-v*n*.xsd cm-featureList.xsd	<Features>	[11]
didl-lite	didl-lite-v*n-yyyymmdd*.xsd didl-lite-v*n*.xsd didl-lite.xsd	<DIDL-Lite>	[15]
dmo	dmo-v*n-yyyymmdd*.xsd dmo-v*n*.xsd dmo.xsd	<DeviceMode>	[16]
dmor	dmor-v*n-yyyymmdd*.xsd dmor-v*n*.xsd dmor.xsd	<DeviceModeRequest>	[17]
dmos	dmos-v*n-yyyymmdd*.xsd dmos-v*n*.xsd dmos.xsd	<DeviceModeStatus>	[18]

Standard Name-space Prefix	Relative URI and File Name [a] • Form 1, Form 2, Form3	Valid Root Element(s)	Schema Reference
pi	pi-v*n*-*yyyymmdd*.xsd pi-v*n*.xsd pi.xsd	<PermissionsInfo>	[20]
rcs-event	rcs-event-v*n*-*yyyymmdd*.xsd rcs-event-v*n*.xsd rcs-event.xsd	<Event>	[22]
rii	rii-*vn-yyyymmdd*.xsd rii-v*n*.xsd rii.xsd	<rendererInfo>	[23]
rpl	rpl-*vn-yyyymmdd*.xsd rpl-v*n*.xsd rpl.xsd	<PlaylistInfo>	[24]
trs	TransformSettings-*vn-yyyymmdd*.xsd TransformSettings-v*n*.xsd TransformSettings.xsd	<TransformSettings>	[28]
srs	srs-v*n-yyyymmdd*.xsd srs-v*n*.xsd srs.xsd	<srs>	[26]
srs-event	srs-event-v*n*-*yyyymmdd*.xsd srs-event-v*n*.xsd srs-event.xsd	<StateEvent>	[27]
upnp	upnp-v*n-yyyymmdd*.xsd upnp-v*n*.xsd upnp.xsd	*n/a*	[29]
Externally Defined Namespaces			
dc	*Absolute URL*: http://dublincore.org/schemas/xmls/simpledc20021212.xsd		[12]
xsd	*n/a*	<schema>	[35]
xsi	*n/a*		*n/a*
xml	*n/a*		[31]

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

4.3.1 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 shall use the standard namespace prefixes as declared in Table 3. In order to properly process the XML documents described herein, control points and devices shall use namespace-aware XML processors [32] for both reading and writing. As allowed by [32], 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 shall 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. However, 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 5.

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

Table 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

4.3.2 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 complies with one or more specific XML schemas, which define XML namespaces. Each namespace is uniquely identified by an assigned namespace name. The namespace names that are defined by the AV Working Committee are URNs. See Table 3 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 qualify and disambiguate 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 subclause 4.3.3).

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

Whenever a new namespace version is created, a new XML schema document (.xsd) is created and published so that the new namespace definition is represented in a machine-readable form. Since a XML schema document is just a representation of a namespace definition, translation errors can occur. Therefore, it is sometime necessary to re-release a published schema in order to correct typos or other namespace representation errors. In order to easily identify the potential multiplicity of schema releases for the same namespace, the URI of each released schema shall 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 4 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, shall use Form 3.
- All UPnP AV published schemas that reference other UPnP AV schemas shall 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 shall contain a `version` attribute in the `<schema>` root element. Its value shall 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"`.

4.3.3 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>
```

4.4 Vendor-defined Extensions

Whenever vendors create additional vendor-defined state variables, actions or properties, their assigned names and XML representation shall follow the naming conventions and XML rules as specified below in subclauses 4.4.1 to 4.4.4.

4.4.1 Vendor-defined Action Names

Vendor-defined action names shall begin with "X_". Additionally, it should be followed by an ICANN assigned domain name owned by the vendor followed by the underscore character ("_"). It shall then be followed by the vendor-assigned action name. The vendor-assigned action name shall 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 shall 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 shall 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 shall not be "XML" in any combination of case.

4.4.2 Vendor-defined State Variable Names

Vendor-defined state variable names shall begin with "X_". Additionally, it should be followed by an ICANN assigned domain name owned by the vendor, followed by the underscore character ("_"). It shall then be followed by the vendor-assigned state variable name. The vendor-assigned state variable name shall not contain a hyphen character ("-", 2D Hex in UTF-8). Vendor-assigned action names are case sensitive. The first character of the name shall 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 shall 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 shall not be "XML" in any combination of case.

4.4.3 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 shall be scoped by a vendor-owned XML namespace. Arbitrary XML shall be enclosed in an element that begins with "X_," and this element shall 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_".

4.4.4 Vendor-defined Property Names

UPnP vendors may add non-standard properties to the ContentDirectory service. Each property addition shall be scoped by a vendor-owned namespace. The vendor-assigned property name shall not contain a hyphen character ("-", 2D Hex in UTF-8). Vendor-assigned property names are case sensitive. The first character of the name shall 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 shall 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 shall not be "XML" in any combination of case.

5 Service Modeling Definitions

5.1 ServiceType

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

urn:schemas-upnp-org:service:*ConnectionManager:3*

5.2 State Variables

5.2.1 State Variable Overview

Table 6 — State Variables

Variable Name	R/A [a]	Data Type	Allowed Value	Default Value	Eng. Units
SourceProtocolInfo	*R*	**string**	CSV [b] (**string**) See 5.2.2		
SinkProtocolInfo	*R*	**string**	CSV (**string**) See 5.2.3		
CurrentConnectionIDs	*R*	**string**	CSV (**ui4**) See 5.2.4		
FeatureList	*R*	**string**	*Features XML Document* See 5.2.5		
ClockUpdateID	*CR* [c]	**ui4**	See 5.2.6		
DeviceClockInfoUpdates	*CR* [c]	**string**	*DeviceClockInfoUpdates XML Document* See 5.2.7		
A_ARG_TYPE_ConnectionStatus	*R*	**string**	See 5.2.8		
A_ARG_TYPE_ConnectionManager	*R*	**string**	See 5.2.9		
A_ARG_TYPE_Direction	*CR* [c]	**string**	See 5.2.10		
A_ARG_TYPE_ProtocolInfo	*R*	**string**	See 5.2.11		
A_ARG_TYPE_ConnectionID	*R*	**i4**	See 5.2.12		
A_ARG_TYPE_AVTransportID	*R*	**i4**	See 5.2.13		
A_ARG_TYPE_RcsID	*R*	**i4**	See 5.2.14		
A_ARG_TYPE_ItemInfoFilter	*CR* [c]	**string**	CSV (**string**) See 5.2.15		
A_ARG_TYPE_Result	*CR* [c]	**string**	See 5.2.16		
A_ARG_TYPE_RenderingInfoList	*CR* [c]	**string**	See 5.2.17		
Non-standard state variables implemented by a UPnP vendor go here	*X*	*TBD*	*TBD*	*TBD*	*TBD*

Note: State variables *A_ARG_TYPE_ConnectionID*, *A_ARG_TYPE_AVTransportID*, and *A_ARG_TYPE_RcsID* are specified as being of data type **i4** to accommodate the fact that some actions require these arguments to contain the special value -1. This special value is used as a return value to indicate that the service is not implemented on the device or is not needed for a particular connection. It can also be used as an *InstanceID* input argument when the actual *InstanceID* value is not (yet) known or the service does not exist.

Action *GetCurrentConnectionIDs()* in this specification and all *InstanceID*s in other services (AVTransport service, RenderingControl service, ...) use data type **ui4** to specify *InstanceID* variables. However, this does not present a problem since a valid *InstanceID* value is always a non-negative integer and is always generated through an argument that is of type **i4**, effectively limiting the valid range for any *InstanceID* to [0, 2^{31}-1]. This range always fits in the valid range of an argument of data type **ui4** (range is [0, 2^{32}-1]) so that an 'out-of-range' error will never occur during assignment.

a For a device this column indicates whether the state variable shall be implemented or not, where *R* = required, *A* = allowed, *CR* = conditionally required, *CA* = conditionally allowed, *X* = Non-standard, add *-D* when deprecated (e.g., *R-D*, *A-D*).

b CSV stands for Comma-Separated Value list. The type between brackets denotes the UPnP data type used for the elements inside the list. The CSV list concept is defined more formally in the ContentDirectory service template.

c See referenced subclause for conditions under which the implementation of this state variable is required.

5.2.2 *SourceProtocolInfo*

This required state variable contains a Comma-Separated Value (CSV) list of information on protocols this ConnectionManager supports for 'sourcing' (sending) data, in its current state.

(The content of the CSV list can change over time, for example due to local resource restrictions on the device.) Besides the traditional notion of the term 'protocol', the protocol-related information provided by the connection also contains other information such as supported content formats. See Annex C for a general discussion on the notion of protocol info. See the table in Annex C.2 for specific allowed values for this state variable. If the device does not support sourcing data, this state variable shall be set to the empty string.

During normal operation, a MediaServer should ensure that there is consistency between what is reported in the *SourceProtocolInfo* state variable and all the *res@protocolInfo* properties of the items that populate the ContentDirectory; that is: at least all protocols that are used by any of the content items should be enumerated in the *SourceProtocolInfo* state variable. (Wildcards ("*") can be used in *SourceProtocolInfo* to limit the number of entries in the CSV list.) Additional protocols that are supported by the MediaServer but are not currently used by any of the content items may also be listed.

Control points can use the *SourceProtocolInfo* CSV list to quickly find out what type of content this MediaServer is capable of serving to the network.

5.2.3 *SinkProtocolInfo*

This required state variable contains a Comma-Separated Value (CSV) list of information on protocols this ConnectionManager supports for 'sinking' (receiving) data, in its current state. (The content of the CSV list can change over time, for example due to local resource restrictions on the device.) The format and allowed value list are the same as for the *SourceProtocolInfo* state variable. If the device does not support 'sinking' data, this state variable shall be set to the empty string.

A MediaRenderer can report temporary unavailability of a protocol (for example, codec not available) by removing the appropriate entries from the *SinkProtocolInfo* CSV list.

5.2.4 *CurrentConnectionIDs*

This required state variable contains a Comma-Separated Value list of references to current active Connections. This list may change without explicit actions invoked by control points, for example by out-of-band cleanup or termination of finished connections.

If allowed action *PrepareForConnection()* is not implemented then this state variable shall be set to "0", indicating that this ConnectionManager service only supports a single connection identified by ConnectionID = 0.

5.2.5 *FeatureList*

This required state variable enumerates the *CM features* (see Annex B) supported by this ConnectionManager service. The value is a valid *Features XML Document*, according to [11].

- The root element of the document is `<Features>`. It contains zero or more child `<Feature>` elements, each of which represents one ConnectionManager service feature that is supported in this implementation.
- A `<Feature>` element shall have a `version` attribute and shall have a `name` attribute containing the assigned name of the feature.
- A `<Feature>` element may have other attributes defined per each feature.

See the schema in [11] for more details on the structure.

5.2.6 *ClockUpdateID*

This conditionally required state variable shall be supported if the *CLOCKSYNC feature* is implemented. It is used to identify the current instance of the *CLOCKSYNC feature*. This state variable is modified whenever a change occurs in the *CLOCKSYNC feature* of the device. A change can be an addition or modification to the `<DeviceClockInfo>` element of the *CLOCKSYNC feature.* The *ClockUpdateID* state variable contains a numeric value that is incremented whenever change occurs in *CLOCKSYNC feature* of the device. Initial value of *ClockUpdateID* state variable shall be zero (0).

5.2.7 DeviceClockInfoUpdates

This conditionally required state variable shall be supported if the *CLOCKSYNC feature* is implemented. It contains a *DeviceClockInfoUpdates XML Document* identifying changes to ClockUpdateID state variable that have occurred since the last time the DeviceClockInfoUpdates state variable was evented. The description of each feature is contained within a child-element of the (one and only one) `<DeviceClockInfoUpdates>` root element. The optional XML header `<?xml version="1.0" ?>` may precede the root element.

The following example shows a "template" for the format of the DeviceClockInfoUpdates state variable. Additional elements and/or attributes may be added to future versions of this specification. Furthermore, a 3rd-party vendor may add vendor-defined elements or attributes.

In order to eliminate element or attribute naming conflicts, the name of any vendor-defined element or attribute shall follow the rules set forth in subclause 4.4.3. All control points should gracefully ignore any element or attribute that it does not understand.

The following XML template includes the *vendor* character style which shows the fields that need to be filled out by individual implementations.

```
<?xml version="1.0" encoding="utf-8"?>
<DeviceClockInfoUpdates
 xmlns="urn:schemas-upnp-org:av:cm-deviceClockInfoUpdates"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="
   urn:schemas-upnp-org:av:cm-deviceClockInfoUpdates
    http://www.upnp.org/schemas/av/cm-deviceClockInfoUpdates.xsd">
   <DeviceClockInfoUpdate
         id="unique ID for this deviceClock instance"
         updateID="ClockUpdateID for this device"/>
   …
</DeviceClockInfoUpdates>
```

`<xml>`

Allowed. Case sensitive.

`<DeviceClockInfoUpdates>`

Required. Shall include a namespace declaration for the ConnectionManager service Feature List Schema ("urn:schemas-upnp-org:av:cm-deviceClockInfoUpdates"). Vendor-defined attributes may be used with the `<deviceClockInfoUpdates>` element. Vendor-defined elements are may be appear as children of the `<deviceClockInfoUpdates>` element. This element shall include one or more of the following elements. This namespace defines the following elements and attributes:

`<DeviceClockInfoUpdate>`

Required. This element includes the ClockUpdateID state variable identifying changes to the *CLOCKSYNC feature*. Vendor-defined attributes may be used with the `<deviceClockInfoUpdate>` element. Vendor-defined elements are may be appear as children of the `<deviceClockInfoUpdate>` element.

Contains the following attributes:

`id`

Required. xsd:string. The ID of for the instance of the `<deviceClockInfo>` element which has been updated (see B.2, Requirements for the *CLOCKSYNC* feature, Version 1).

`updateID`

Required. xsd:unsignedInt. Contains the value of the ClockUpdateID state variable that resulted when the `<deviceClockInfo>` element was added or modified.

Example:

The following example shows update information for three different clock devices.

```
<?xml version="1.0" encoding="utf-8"?>
<deviceClockInfoUpdates
   xmlns="urn:schemas-upnp-org:av:cm-deviceClockInfoUpdates"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
   xsi:schemaLocation="
      urn:schemas-upnp-org:av:cm-deviceClockInfoUpdates
      http://www.upnp.org/schemas/av/cm-deviceClockInfoUpdates.xsd">
   <deviceClockInfoUpdate id="Clock#1" updateID="1" />
   <deviceClockInfoUpdate id="Clock#2" updateID="1" />
   <deviceClockInfoUpdate id="Clock#3" updateID="3" />
</deviceClockInfoUpdates>
```

5.2.8 A_ARG_TYPE_ConnectionStatus

This required state variable is introduced to provide type information for the Status argument in the GetCurrentConnectionInfo() action. This status may change dynamically due to changes in the network.

Table 7 — allowedValueList for A_ARG_TYPE_ConnectionStatus

Value	R/A
"OK"	R
"ContentFormatMismatch"	R
"InsufficienBandwidth"	R
"UnreliableChannel"	R
"Unknown"	R
Vendor-defined	X

5.2.9 A_ARG_TYPE_ConnectionManager

This required state variable is introduced to provide type information for the PeerConnectionManager argument in actions PrepareForConnection() and GetCurrentConnectionInfo(). A ConnectionManager reference takes the form of a **UDN**/**serviceId** pair (the slash is the delimiter). A control point can use UPnP discovery (SSDP) to obtain a ConnectionManager's description document from the UDN. Subsequently, the ConnectionManager's service description can be obtained by using the **serviceId** part of the reference.

5.2.10 A_ARG_TYPE_Direction

This conditionally required state variable shall be supported when the PrepareForConnection() action is implemented. It is used to provide type information for the Direction argument in the PrepareForConnection() action.

Table 8 — allowedValueList for A_ARG_TYPE_Direction

Value	R/A
"Output"	R
"Input"	R

5.2.11 A_ARG_TYPE_ProtocolInfo

This required state variable is introduced to provide type information for the RemoteProtocolInfo argument in action PrepareForConnection() and the ProtocolInfo argument in action GetCurrentConnectionInfo().

5.2.12 A_ARG_TYPE_ConnectionID

This required state variable is introduced to provide type information for the ConnectionID argument in actions PrepareForConnection(), ConnectionComplete() and GetCurrentConnectionInfo().

5.2.13 A_ARG_TYPE_AVTransportID

This required state variable is introduced to provide type information for the AVTransportID argument in actions: PrepareForConnection() and GetCurrentConnectionInfo(). It identifies a

logical instance of the AVTransport service associated with a Connection. See AVTransport specification [5], Annex B.6 for more information.

5.2.14 A_ARG_TYPE_RcsID

This required state variable is introduced to provide type information for the RcsID argument in actions PrepareForConnection() and GetCurrentConnectionInfo(). It identifies a logical instance of the RenderingControl service associated with a connection. See RenderingControl specification [21], subclause 1.2 and Annex A.1 for more information.

5.2.15 A_ARG_TYPE_ItemInfoFilter

This conditionally required state variable shall be supported when the GetRandererItemInfo() action is implemented. It is introduced to provide type information for the ItemInfoFilter argument in the GetRendererItemInfo() action. The ItemInfoFilter argument applies to a document using the schema [23] as the default namespace. The comma-separated list of property specifiers indicates which metadata properties are to be returned in the ItemRenderingInfo argument of the GetRendererItemInfo() action. Each property name shall include the predefined namespace prefix for that property, except for the “rii:” namespace prefix which is assumed by default. All dependent property names shall be fully qualified using the double colon (“::”).

For example:

```
itemInfo::resPlaybackInfo::drmInfo
```

The ItemInfoFilter argument allows control points to control the complexity of the object metadata properties that are returned within the ItemRenderingInfo argument of the GetRendererItemInfo() action. Properties required by the schema [23] are always returned in the ItemRenderingInfo output argument. The ItemInfoFilter argument allows a control point to specify additional properties, not required by the schema [23] to be returned in ItemRenderingInfo. The GetRenderingItemInfo() action should not return optional properties unless they are explicitly requested in the ItemInfoFilter input argument.

Both independent and dependent properties may be included in the comma-separated ItemInfoFilter argument. If the ItemInfoFilter argument is equal to “*”, all supported properties, both required and allowed, from all namespaces are returned. An independent property or an independent child property may be suffixed by the “#” U+0023 character. When present, this suffix indicates that the actions associated with the A_ARG_TYPE_ItemInfoFilter argument shall return all child properties descended from the indicated property.

The GetRenderingItemInfo() action shall also ignore optional properties requested in the ItemInfoFilter input argument, which are not actually present in the matching objects.

The A_ARG_TYPE_ItemInfoFilter state variable is intended to follow semantics similar to those defined for the A_ARG_TYPE_Filter state variable in the ContentDirectory service.

Example 1: The ItemInfoFilter argument in a GetRendererItemInfo() action is specified as `itemInfo::resPlaybackInfo::drmInfo#`, indicating that the action should return digital rights information for each res property.

Request:

```
GetRendererItemInfo(
"itemInfo::resPlaybackInfo::drmInfo#",
"<?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="Item_0002" parentID="13" restricted="0">
      <dc:title>Try a little tenderness</dc:title>
      <upnp:class>object.item.audioItem.musicTrack</upnp:class>
      <res protocolInfo="http-get:*:audio/mpeg:*" size="3558000">
         http://168.192.1.1/audio197.mp3
      </res>
   </item>
</DIDL-Lite>" )
```

Response:

```
GetRendererItemInfo(
"<rendererInfo xmlns="urn:schemas-upnp-org:av:rii"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="
      urn:schemas-upnp-org:av:rii
      http://www.upnp.org/schemas/av/rii.xsd">
   <itemInfo itemID="Item_0002">
      <resPlaybackInfo resIndex="0" canPlay="1">
         <drmInfo drmProtected="1" drmStatus="OK">
            <drmSystem>
               <friendlyName>Not So Free Music</friendlyName>
                <ICANNName>OpenMobileAlliance.ORG</ICANNName>
               <systemName>DRM_REL_DD</systemName>
               <systemVersion>2.1</systemVersion>
            </drmSystem>
            <licenseRights type="play">
               <licenseUsageTimeRemaining>
                  P5D20:00:00
               </licenseUsageTimeRemaining>
               <licenseSubscriptionTimeRemaining>
                  P1D20:00:00
               </licenseSubscriptionTimeRemaining>
               <licenseUsageCountRemaining>4</licenseUsageCountRemaining>
            </licenseRights>
         </drmInfo>
      </resPlaybackInfo>
   </itemInfo>
</rendererInfo>" )
```

5.2.16 *A_ARG_TYPE_Result*

This conditionally required state variable shall be supported when the *GetRendererItemInfo()* action is implemented. It is introduced to provide type information for the *Result* argument in various actions. The structure of the *Result* argument is a *DIDL-Lite XML Document*:

- Optional XML declaration `<?xml version="1.0" ?>`
- `<DIDL-Lite>` is the root element.
- `<item>` is the element representing objects of class *item* and all its derived classes.
- Elements in the Dublin Core (“dc:”) and UPnP (“upnp:”) namespaces represent object metadata.
- See the DIDL-Lite schema [15] for more details on the structure. The available properties and their names are described in ContentDirectory specification [7], Annex B.

Note that since the value of *Result* is XML, it needs to be escaped (using the normal XML rules: [36], and [37], 2.4 Character Data and Markup) before embedding in a SOAP response message. In addition, when a value of type *A_ARG_TYPE_Result* is employed in a CSV list, commas (“,”) that appear within XML CDATA shall be escaped as “\,”. See subclause 4.1.2.

5.2.17 A_ARG_TYPE_RenderingInfoList

This conditionally required state variable shall be supported when the GetRendererItemInfo() action is implemented. It returns a document described by the schema [23], detailing whether the render can play the indicated item formats. The structure of the document is as follows:

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

   <itemInfo
      itemID="object ID (@id property) of corresponding item">

      <resPlaybackInfo
         resID="res@id for item's res property [if available]"
         resIndex="zero-based index of item's res property "
         canPlay="boolean indicator that res property's content is playable" >

         <drmInfo
            drmProtected="indicates whether this content-binary is DRM protected"
            drmStatus="indicator of DRM status for this res property">

            <drmSystem>
               <friendlyName>informative name of DRM system</friendlyName>
               <ICANNName>ICANN name of DRM system provider</ICANNName>
               <systemIdentifier>
                  normative name of DRM system
               </systemIdentifier>
               <systemVersion>
                  normative version of DRM system
               </systemVersion>
            </drmSystem>

            <licenseIdentifier>
               String identifying the DRM license instance.
            </licenseIdentifier>

            <licenseRights type ="play | copy | store | stream">

               <licenseSubscriptionTimeRemaining>
                   A duration indicating the amount of time remaining that the
                   end-user is allowed to begin using the license-right indicated
                   by   the "type=" attribute or the <licenseRights> element.
               </licenseSubscriptionTimeRemaining>

               <licenseValidTimeRemaining>
                  A duration indicating the amount of time remaining that the
                  end-user is allowed to use the license right indicated by the
                  "type=" attribute or the <licenseRights> element.
               </licenseValidTimeRemaining>

               <licenseUsageTimeRemaining>
                  A duration indicating the amount of cumulative usage time
                  remaining that the end-user is allowed to use the license right
                  indicated by the "type=" attribute or the <licenseRights>
                  element.
               </licenseUsageTimeRemaining>

               <licenseUsageCountRemaining>
                  An integer indicating the number of times remaining that the
                  end-user may use the license right indicated by the "type="
```

```
                  attribute or the <licenseRights> element.
               </licenseUsageCountRemaining>

            </licenseRights>

         </drmInfo>

         <playbackInfo
             playbackCompatibility="A status code indicating the device's ability
             to   play this content" />

         <videoStreamInfo
            outputResolution="Expected playback resolution" />

         <audioStreamInfo
            outputChannels="Expected number of audio output channels" />

         <imageStreamInfo
            outputResolution="Expected image output resolution" />

         </playbackInfo>

         <transformInfo>
            <TransformList
                  xmlns="urn:schemas-upnp-org:av:AllowedTransformSettings"
                  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                  xsi:schemaLocation=
                     "urn:schemas-upnp-org:av:AllowedTransformSettings
                     http://www.upnp.org/schemas/av/AllowedTransformSettings.xsd">
               <transform name="Name of transform"
                        shared="1 or 0 indicating shared ">
                  <friendlyName>Friendly name of transform</friendlyName>
                  <allowedValueRange
                     unit="unit of the transform"
                     scale="scale indication of the transform"
                      inactiveValue="value of the transform that does not result
                      in any effect on the (original) output">
                     <minimum>minimum value</minimum>
                     <maximum>maximum value</maximum>
                     <step>increment value</step>
                  </allowedValueRange>
               </transform>
               <transform name="Name of transform"
                        shared="1 or 0 indicating shared">
                  <friendlyName>Friendly name of transform</friendlyName>
                  <allowedValueList
                     unit="unit of the transform"
                     scale="scale indication of the transform"
                      inactiveValue="value of the transform that does not result
                      in any effect on the (original) output">
                     <allowedValue>enumerated value</allowedValue>
                     <!-- other allowed values go here -->
                  </allowedValueList>
               </transform>
               <!-- other transforms go here -->
            </TransformList>
         </transformInfo>
      </resPlaybackInfo>
      … Additional <resPlaybackInfo> elements [one per item <res> property] …

   </itemInfo>

   … Additional <itemInfo> [one per <item> ] …

</rendererInfo>
```

?xml

Allowed. Case sensitive.

rendererInfo

Required. Shall include a namespace declaration for the schema (“urn:schemas-upnp-org:av:rii”) [23] and shall include zero or more of the following elements.

itemInfo

Required. <XML>, Provides a description of the renderer’s ability to interpret and render a ContentDirectory service item. It consist of a series of dependent elements that describe various aspects of the Rendering device capabilities.

Note, each occurrence of this element corresponds to a ContentDirectory service item. If no ContentDirectory service items are found, then this element shall be omitted.

@itemID

Required. xsd:string, For each <itemInfo> element the itemID attribute contains the value of the *@id* property of a corresponding ContentDirectory service item.

resPlaybackInfo

Required. <XML>, Provides an overall status of the Renderer’s ability to play the content-binary identified by the resIndex attribute of this element. In addition, the <resPlaybackInfo> element includes dependent elements that describe specific characteristics of the content-binary.

Note, each occurrence of this element corresponds to a *<res>* element within the ContentDirectory service item. If no *<res>* properties are found in the ContentDirectory service item then this element shall be omitted.

@resID

Required. xsd:string, Returns the *res@id* property of the *<res>* element within the ContentDirectory service item identified by the itemID attribute of the <itemInfo> element. If the ContentDirectory service item *res@id* property is not present the “” value shall be returned for this attribute.

@resIndex

Required. xsd:unsignedInt, The index of the *res* property within the ContentDirectory service item identified by the itemID attribute of the <itemInfo> element.

@canPlay

Required. xsd:boolean, Indicates whether the rendering device expects to be able to play the content-binary identified by the resIndex attribute of the <resPlaybackInfo> element.

drmInfo

Allowed. <XML>, Provides an overall indication of the DRM status for the content-binary identified by the resIndex attribute of the <resPlaybackInfo> parent element. In addition, this element may contain detailed DRM system related information.

@drmProtected

Required. xsd:boolean, This property indicates whether the content-binary identified by the resIndex attribute is DRM protected.

@drmStatus

Required. xsd:string, This property indicates the overall DRM status for the content-binary identified by the resIndex attribute. Allowed values for the drmStatus attribute are defined in the AVTransport specification [5]. See state variable *DRMState*.

drmSystem

Allowed. <XML>, Identifies the DRM system used to protect the content-binary identified by the resIndex attribute.

friendlyName

Allowed. xsd:string Provides an informative name for the DRM system used to protect the content-binary identified by the resIndex attribute. The value of this element is specified by the device implementer.

ICANNName

Required. xsd:string, Identifies the ICANN name for the DRM system used to protect the content-binary identified by the resIndex attribute. The value of this element is specified by the organization that defined the DRM system.

systemIdentifier

Allowed. xsd:string, Identifies the normative name for the DRM system used to protect the content-binary identified by the resIndex attribute. The value of this element is specified by the organization that defined the DRM system.

systemVersion

Allowed. xsd:string, Identifies the version for the DRM system used to protect the content-binary identified by the resIndex attribute. The value of this element is specified by the organization that defined the DRM system.

licenseIdentifier

Allowed, xsd:string, Identifies the DRM license instance for this content binary. The contents of this field are vendor defined by the DRM system used to protect the content-binary identified by the resIndex attribute.

licenseRights

Allowed, <XML>, Indicates the types of rendering operations this device is permitted to perform by the DRM system for the content-binary identified by the resIndex attribute. The licenseRights element may appear multiple times to provide license information for each license "right" conveyed by the content provider.

@type

Required,xsd:string, Indicates the type of right the content-binary license provides. The following license rights are defined:

"play"

Indicates that the content-binary license identified by the resIndex attribute allows local playback (but not necessarily storage, copying or streaming of this item).

"store"

Indicates that the content-binary license identified by the resIndex attribute allows making of an internal copy of this DRM protected item.

"copy"

Indicates that the content-binary license identified by the resIndex attribute allows making of an external copy of this DRM protected item.

"stream"

Indicates that the content-binary license identified by the resIndex attribute allows streaming of this DRM protected item over an encrypted link to an external device.

licenseUsageTimeRemaining

Allowed. xsd:string, Indicates the amount of cumulative playback time (duration) remaining on the current DRM license. This playback time may be accumulated separate playback sessions. This value is required to conform to the EBNF defined in ContentDirectory service [7], Annex E.1.

For example: A video rental license may be available for 1 week but may only allow a 24-hour viewing period. In this case, the <licensePlaybackTime> element value would be set to 24-hours and begins to countdown when the content is first accessed. When this element value counts down to zero, the content would no longer be available regardless of whether the <licenseSubscriptionTimeRemaining> element indicated additional time is available.

licenseValidTimeRemaining

Allowed. xsd:string, Indicates the amount of "wall-clock" time duration that the current DRM license will remain valid. This value is required to conform to the EBNF defined in ContentDirectory service [7], Annex E.1.

licenseSubscriptionTimeRemaining

Allowed. xsd:string, Indicates the amount of time remaining that the duration content license will be available to the end-user. This value is required to comform to the EBNF defined in ContentDirectory service [7], Annex E.1. For example: A video rental license may be available

for 1 week but may only allow a 24-hour viewing period. In this case, the <licenseSubscriptionTimeRemaining> element value would reflect the amount of time remaining in the 7-day availability period of the content license.

licenseUsageCountRemaining

Allowed. xsd:unsignedInt, This element indicates the number of times remaining that the end-user use this license right. This element should be omitted of there is no count restriction on the current license right.

playbackInfo

Allowed. <XML>, Provides an overall indication of the rendering device's ability to play the content-binary identified by the resIndex attribute of the <resPlaybackInfo> parent element. In addition, this element may contain detailed information concerning the rendering device's capabilities to render the content.

@playbackCompatibility

Required. xsd:string, This property indicates the overall playback error state for the content binary. If the content cannot be played due to playback format or resource related issues, then this field indicates the playback related error codes.

The following Playback error codes are defined:

"*OK*"

The rendering device expects to be able to render this content.

"*DRM_ERROR*"

The rendering device has detected a DRM related error condition that is likely to prevent playback of this content-binary.

"*OTHER_PLAYBACK_ERROR*"

The rendering device has detected some other condition (not described by other defined Playback error codes) that is likely to prevent playback of this content-binary.

"*MEDIA_FORMAT_NOT_SUPPORTED*"

The rendering device has determined that the media format for this content is not supported by the rendering device.

"*MEDIA_FORMAT_EXCEEDS_DEVICE_CAPABILITIES*"

The rendering device indicates that the format for this content-binary is understood, however, it is likely that this content-binary will not play successfully. Examples would include a JPEG image that is too large to process, or an audio item that exceeds the maximum bitrate the renderer can support.

"*MULTI_STREAM_PLAYBACK_NOT_SUPPORTED*"

The content binary has additional metadata indicating that it has one or more secondary binaries associated with it, which shall be played together in a synchronized way. The rendering device indicates that it is not capable of playing these content binaries simultaneously.

videoStreamInfo

Allowed. <XML>, Provides detailed information about the rendering device presentation of the video stream portion of the content-binary identified by the resIndex attribute of the <resPlaybackInfo> parent element.

@outputResolution

Required. xsd:string, This element indicates the resolution of the output stream of the renderer. The output resolution returned should reflect the effective resolution presented to the end-user based on the renderer's current configuration. The format of this field shall be as follows:

video-width 'x' video-height ('i' | 'p') [['/'] video-frame-rate] [',' vendor-specific]

For example:

```
1920x1080i
1280x720p60
720x576p/25
640x480i60,param1=10,param2=30
```

audioStreamInfo

Allowed. <XML>, Provides detailed information about the rendering device presentation of the audio stream portion of the content-binary identified by the resIndex attribute of the <resPlaybackInfo> parent element.

@outputChannels

Required. xsd:string, This element describes the number of separate channels the renderer will provide to the end-user as currently configured. This includes channels that the render creates locally by "expanding" the input audio stream. Conversely, it also may indicate that the renderer is configured to (or capable of) providing fewer channels than provided in the input stream. The field is encoded as a speaker channel configuration, either "n.n" or "n", where the 1st integer indicates the number of full-range of channels and the 2nd integer indicates the number of low-frequence channels.

For example:

```
7.1
5.1
2
```

imageStreamInfo

Allowed. <XML>, The <imageStreamInfo> element provides detailed information about the rendering device's presentation of an image content-binary image (ex: JPEG, PNG) identified by the resIndex attribute of the <resPlaybackInfo> parent element.

@outputResolution

Required. xsd:string, This element indicates the resolution of the image as presented to the end-user using the renderer's default settings. The field is encoded as a WxH resolution (640x480).

transformInfo

Allowed. <XML>, Provides a list of content transforms that are within the rendering device's ability to be performed on the content-binary identified by the resIndex attribute of the <resPlaybackInfo> parent element. The one and only child element of this XML element is an *AllowedTransformSettings XML Document*, defined by the *AllowedTransformSettings* state variable in the RenderingControl service specification [21]. The list of transforms for the identified content-binary shall be the same as if it is returned on an invocation of the *RenderingControl::GetAllowedTransformSettings()* action using an instanceID to which this content-binary's resource is bound.

5.3 Eventing and Moderation

Table 9 — Event Moderation

Variable Name	Evented	Moderated Event	Max Event Rate [a] (seconds)	Logical Combination	Min Delta per Event [b]
SourceProtocolInfo	*YES*	*NO*			
SinkProtocolInfo	*YES*	*NO*			
CurrentConnectionIDs	*YES*	*NO*			
FeatureList	*NO*	*NO*			
ClockUpdateID	*NO*	*NO*			
DeviceClockInfoUpdates	*YES*	*NO*			
Non-standard state variables implemented by a UPnP vendor go here	*TBD*	*TBD*	*TBD*	*TBD*	*TBD*

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

b (N) * (allowedValueRange Step).

5.4 Actions

5.4.1 Action Overview

Immediately following this table is detailed information about these actions, including short descriptions of the actions, the effects of the actions on state variables, and error codes defined by the actions.

Table 10 — Actions

Name	R/A [a]	Control Point R/A [b]
GetProtocolInfo()	R	CR [c]
PrepareForConnection()	A	A [d]
ConnectionComplete()	A	A
GetCurrentConnectionIDs()	R	A
GetCurrentConnectionInfo()	R	A
GetRendererItemInfo()	A	A
GetFeatureList()	R	R
Non-standard actions implemented by a UPnP vendor go here	X	X

a For a device this column indicates whether the state variable shall be implemented or not, where R = required, A = allowed, CR = conditionally required, CA = conditionally allowed, X = Non-standard, add -D when deprecated (e.g., R-D, A-D).

b For a control point this column indicates whether a control point shall be capable of invoking this action, where R = required, A = allowed, CR = conditionally required, CA = conditionally allowed, X = Non-standard, add -D when deprecated (e.g., R-D, A-D).

c Only required for a MediaRenderer control point (which is part of a 3-box scenario).

d The control point is not required to call the PrepareForConnection() action when implemented in the device, but when the control point does not use the PrepareForConnection() the interaction with the device will always be with an instanceID value of 0 and therefore will use less functionality of the MediaRenderer device.

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

5.4.2 GetProtocolInfo()

This required action returns the protocol-related info that this ConnectionManager supports in its current state, as a Comma-Separated Value list of strings according to Table C.1. Protocol-related information for ‘sourcing’ data is returned in the Source argument and protocol-related information for ‘sinking’ data is returned in the Sink argument. When this ConnectionManager resides in a device that only supports ‘sourcing’ of data, the Sink argument shall return the empty string. Likewise, when this ConnectionManager resides in a device that only supports ‘sinking’ of data, the Source argument shall return the empty string.

5.4.2.1 Arguments

Table 11 — Arguments for GetProtocolInfo()

Argument	Direction	relatedStateVariable
Source	OUT	SourceProtocolInfo
Sink	OUT	SinkProtocolInfo

5.4.2.2 Dependency on State

None.

5.4.2.3 Effect on State

None.

5.4.2.4 Errors

None

5.4.3 *PrepareForConnection()*

This allowed action is used to allow the device to prepare itself to connect to the network for the purposes of sending or receiving media content (for example, a video stream). *PrepareForConnection()* also allows the device to indicate whether or not it can establish a connection based on the current status of the device and/or the current conditions of the network.

The *RemoteProtocolInfo* input argument identifies the protocol, network, and format that shall be used to transfer the content.

- If *PrepareForConnection()* is invoked on a MediaServer device, the *RemoteProtocolInfo* argument shall be set to one of the *ProtocolInfo* entries from the CSV list obtained from the peer MediaRenderer device via the *GetProtocolInfo()* action (see Annex C.2 for details). If the peer device does not implement *GetProtocolInfo()* (because it is not a MediaRenderer or not even a UPnP device), then the *RemoteProtocolInfo* argument shall be set to one of the *ProtocolInfo* entries returned by the *GetProtocolInfo()* action on the local MediaServer device.
- If *PrepareForConnection()* is invoked on a MediaRenderer device, the *RemoteProtocolInfo* argument shall be set to the value of the *protocolInfo* attribute of the content item (located in the ContentDirectory on the peer MediaServer device) that is going to be played (see Annex C.2 for details). If the peer device does not implement a ContentDirectory service (because it is not a MediaServer or not even a UPnP device), then the *RemoteProtocolInfo* argument shall be set to one of the *ProtocolInfo* entries returned by the *GetProtocolInfo()* action on the local MediaRenderer device.

The *ConnectionID* out argument is used to identify the connection that was prepared by the device in response to this invocation. The *ConnectionID* is a device-specific value and is not unique throughout the network. Therefore, the *ConnectionIDs* returned by the two end-points of the same connection will generally not be the same value. Refer to *GetCurrentConnectionIDs()* and/or the UPnP AV Device Architecture document for additional information. The *AVTransportID* and *RcsID* out arguments are used to identify the AVTransport and RenderingControl services that are associated with the connection. The returned values are the *InstanceIDs* that need to be used when invoking subsequent invocations of the AVTransport and RenderingControl services. An *InstanceID* value of -1 indicates the device did not associate an AVTransport and/or RenderingControl service with this connection. The returned *ConnectionID*, *AVTransportID*, and *RcsID* become invalid when the device closes the connection. This will occur when *ConnectionComplete()* is invoked or any other time the device decides to close the connection (a.k.a auto-cleanup). Refer to *ConnectionComplete()* for additional information.

This action is marked allowed which means that each device manufacturer decides whether or not to implement it. Therefore, some devices will implement *PrepareForConnection()* while other devices will not. Since *PrepareForConnection()* allows a device to prepare itself to connect to the network, if a device has implemented that action, control points need to invoke *PrepareForConnection()* before attempting to stream any content; that is: before invoking *AVTransport::SetAVTransportURI()* (see Annex C.3). Otherwise, the device may not operate correctly because it has not been properly configured. Additionally, control points need to invoke *PrepareForConnection()*, if implemented, so that the device can inform the control point, via an error code, that the device's current operating environment is not able to accommodate the requested stream.

Once a connection has been prepared, it can be used to transfer several pieces of the content before calling *ConnectionComplete()* as long as each content item is compatible with the *RemoteProtocolInfo* argument that was passed into *PrepareForConnection()*; that is: each content item has the same media format as specified in *RemoteprotocolInfo*.

If a device does not implement PrepareForConnection(), it shall only support a single connection at any time. This connection is implicitly assumed to be always present and is identified by ConnectionID = 0.

5.4.3.1 Arguments

Table 12 — Arguments for PrepareForConnection()

Argument	Direction	relatedStateVariable
RemoteProtocolInfo	IN	A_ARG_TYPE_ProtocolInfo
PeerConnectionManager	IN	A_ARG_TYPE_ConnectionManager
PeerConnectionID	IN	A_ARG_TYPE_ConnectionID
Direction	IN	A_ARG_TYPE_Direction
ConnectionID	OUT	A_ARG_TYPE_ConnectionID
AVTransportID	OUT	A_ARG_TYPE_AVTransportID
RcsID	OUT	A_ARG_TYPE_RcsID

5.4.3.2 Dependency on State

None.

5.4.3.3 Effect on State

This action prepares the device to stream content to or from the specified peer ConnectionManager, according to the specified direction and protocol information. The PeerConnectionManager input argument identifies the ConnectionManager service on the other side of the connection. The PeerConnectionID input argument identifies the specific connection on that ConnectionManager service. This information allows a control point to *link* a connection on device A to the corresponding connection on device B, via action GetCurrentConnectionInfo(). If the PeerConnectionID is not known by a control point (for example, this is the first of the two PrepareForConnection() actions), or the peer device doesn't implement PrepareForConnection() then this value shall be set to reserved value -1.

This action returns a locally unique ID for the established Connection in the ConnectionID argument, and adds that ConnectionID to state variable CurrentConnectionIDs. This ConnectionID might be used by a control point to manually terminate the established Connection through (allowed) action ConnectionComplete(). It can also be used to retrieve information associated with the Connection via action GetCurrentConnectionInfo(). Value -1 is reserved, and shall not be returned.

This action may return a virtual InstanceID of a local AVTransport service in the AVTransportID argument. This AVTransportID shall be passed as an input argument to the local AVTransport service action invocations. If the returned AVTransportID is -1 (reserved value), then there is no AVTransport service on this device that can be used to control the established connection. This is dependent on the 'push' or 'pull' nature of the streaming protocol.

This action may return a virtual InstanceID of a local RenderingControl service in the RcsID argument. This RcsID shall be passed as an input argument to the local RenderingControl service action invocations. If the returned RcsID is -1 (reserved value), then there is no RenderingControl service on this device, for example, because the device is a source device (MediaServer) rather than a sink device (MediaRenderer).

Due to local restrictions on the device running the ConnectionManager, state variables SourceProtocolInfo and/or SinkProtocolInfo may change (for example, certain physical ports on the device are not available anymore for new connections) as a result of this action.

5.4.3.4 Errors

Table 13 — Error Codes for *PrepareForConnection()*

errorCode	errorDescription	Description
400-499	TBD	See Control clause in [14].
500-5999	TBD	See Control clause in [14].
600-699	TBD	See Control clause in [14].
701	Incompatible protocol info	The connection cannot be established because the protocol info argument is incompatible.
702	Incompatible directions	The connection cannot be established because the directions of the involved ConnectionManagers (source/sink) are incompatible.
703	Insufficient network resources	The connection cannot be established because there are insufficient network resources (bandwidth, channels, etc.).
704	Local restrictions	The connection cannot be established because of local restrictions in the device. This might happen, for example, when physical resources on the device are already in use by other connections.
705	Access denied	The connection cannot be established because the client is not permitted to access the specified ConnectionManager.
707	Not in network	The connection cannot be established because the ConnectionManagers are not part of the same physical network.
708	Connection Table overflow	The connection cannot be established because the specified ConnectionManager has instantiated the maximum number of simultaneous connections it has room for in its internal data structures. Closing one connection will resolve the issue.
709	Internal processing resources exceeded	The connection cannot be established because the device does not have sufficient internal processing resources to handle the new connection. Closing one or more connections on this device may resolve the issue.
710	Internal memory resources exceeded	The connection cannot be established because the device does not have sufficient internal memory resources to handle the new connection. Closing one or more connections on this device may resolve the issue.
711	Internal storage system capabilities exceeded	The connection cannot be established because the device does not have sufficient internal storage system capabilities to handle the new connection. Closing one or more connections on this device may resolve the issue.

5.4.4 *ConnectionComplete()*

This allowed action is used to inform the device that the specified connection, which was previously allocated by *PrepareForConnection()*, is no longer needed. Any resources that were allocated for that connection during *PrepareForConnection()* can be freed by the device at its discretion.

In some situations, *ConnectionComplete()* may never be invoked; for example, the control point spontaneously goes away. In order to prevent an unused connection from permanently consuming resources, the device should automatically cleanup unused connections. The process for determining when a connection should be automatically cleaned up is implementation dependent. For example, a device may decide to close a connection after the connection has been inactive for a certain period of time. Alternatively, a device may decide to close a connection when it needs to free the resources that are associated with the connection. See Annex C.5 for additional information.

5.4.4.1 Arguments

Table 14 — Arguments for *ConnectionComplete()*

Argument	Direction	relatedStateVariable
ConnectionID	*IN*	*A_ARG_TYPE_ConnectionID*

5.4.4.2 Dependency on State

None.

5.4.4.3 Effect on State

This action removes the connection referenced by argument *ConnectionID* by modifying state variable *CurrentConnectionIDs*, and (if necessary) performs any protocol-specific cleanup actions such as releasing network resources. See Annex A for more details.

Due to local restrictions on the device running the ConnectionManager, state variables *SourceProtocolInfo* and/or *SinkProtocolInfo* may change (for example, certain physical ports on the device are freed up for new connections).

5.4.4.4 Errors

Table 15 — Error Codes for *ConnectionComplete()*

errorCode	errorDescription	Description
400-499	TBD	See Control clause in [14].
500-599	TBD	See Control clause in [14].
600-699	TBD	See Control clause in [14].
706	Invalid connection reference	The connection reference argument does not refer to a valid connection established by this service.

5.4.5 *GetCurrentConnectionIDs()*

This required action returns a Comma-Separated Value list of *ConnectionID*s of currently ongoing Connections. A *ConnectionID* can be used to manually terminate a Connection via action *ConnectionComplete()*, or to retrieve additional information about the ongoing Connection via action *GetCurrentConnectionInfo()*.

If a device does not implement *PrepareForConnection()*, this action shall return the single value "0".

5.4.5.1 Arguments

Table 16 — Arguments for *GetCurrentConnectionIDs()*

Argument	Direction	relatedStateVariable
ConnectionIDs	*OUT*	*CurrentConnectionIDs*

5.4.5.2 Dependency on State

None.

5.4.5.3 Effect on State

None.

5.4.5.4 Errors

Table 17 — Error Codes for *GetCurrentConnectionIDs()*

errorCode	errorDescription	Description
400-499	TBD	See Control clause in [14].
500-599	TBD	See Control clause in [14].
600-699	TBD	See Control clause in [14].

5.4.6 *GetCurrentConnectionInfo()*

This required action returns associated information of the connection referred to by the *ConnectionID* input argument. The *AVTransportID* argument may be the reserved value -1 and

the PeerConnectionManager argument may be the empty string in cases where the connection has been setup completely out of band, not involving a PrepareForConnection() action.

If the allowed action PrepareForConnection() is not implemented then (limited) connection information can be retrieved for ConnectionID 0. The device shall return the following information:

- RcsID shall be 0 (a single instance of the RenderingControl service is implemented) or -1 (RenderingControl service is not implemented)
- AVTransportID shall be 0 (a single instance of the AVTransport service is implemented) or -1 (AVTransport service is not implemented)
- ProtocolInfo shall contain accurate information if it is known, otherwise it shall be the empty string
- PeerConnectionManager shall be the empty string
- PeerConnectionID shall be -1
- Direction shall be "Input" or "Output"
- Status shall be "OK" or "Unknown"

5.4.6.1 Arguments

Table 18 — Arguments for GetCurrentConnectionInfo()

Argument	Direction	relatedStateVariable
ConnectionID	IN	A_ARG_TYPE_ConnectionID
RcsID	OUT	A_ARG_TYPE_RcsID
AVTransportID	OUT	A_ARG_TYPE_AVTransportID
ProtocolInfo	OUT	A_ARG_TYPE_ProtocolInfo
PeerConnectionManager	OUT	A_ARG_TYPE_ConnectionManager
PeerConnectionID	OUT	A_ARG_TYPE_ConnectionID
Direction	OUT	A_ARG_TYPE_Direction
Status	OUT	A_ARG_TYPE_ConnectionStatus

5.4.6.2 Dependency on State

None.

5.4.6.3 Effect on State

None.

5.4.6.4 Errors

Table 19 — Error Codes for GetCurrentConnectionInfo()

errorCode	errorDescription	Description
400-499	TBD	See Control clause in [14].
500-599	TBD	See Control clause in [14].
600-699	TBD	See Control clause in [14].
706	Invalid connection reference	The connection reference argument does not refer to a valid connection established by this service.

5.4.7 GetRendererItemInfo()

This allowed action allows the control point to request that the rendering device inspect item metadata provided and determine if the rendering device expects to be able to successfully play the item described. The MediaRenderer control point issuing this action submits one or more items obtained from a MediaServer device using the ContentDirectory service's Browse() or

Search() actions. Each item will normally contain one or more *res* properties that specify "content-binaries" that the MediaRenderer device may play. The *ItemRenderingInfoList* output argument is an XML document using the schema [23] that contains:

a) A root element: `<rendererInfo>`

b) Zero or more `<itemInfo>` child elements. Each `<itemInfo>` element corresponds to an *item* that appears in the *ItemMetadataList* parameter.

c) Within each `<itemInfo>` element, a `<resPlaybackInfo>` child element is generated for each item *res* property. The `<resPlaybackInfo>` element describes the overall capability of the MediaRenderer to play this content-binary. In addition, the `<resPlaybackInfo>` element may optionally contain DRM license information that the MediaRenderer device's DRM agent may provide.

The control point may use the *ItemInfoFilter* argument to control the amount of information returned for the *item* submitted.

5.4.7.1 Arguments

The following list presents an overview of the *GetRendererItemInfo()* action arguments.

- *ItemInfoFilter*: See subclause 5.2.15.
- *ItemMetadataList*: A DIDL-Lite document containing the item metadata for the rendering device to inspect. See subclause 5.2.16. When constructing an ItemMetadataList, it is recommended that items submitted to the MediaRenderer device by this action include all available item properties. This would be done by using a *Filter* argument of "*" on the ContentDirectory service's *Browse()* or *Search()* actions. Following this recommendation ensures that the MediaRenderer device can examine any item metadata that it finds relevant.
- *ItemRenderingInfoList*: An XML document containing rendering information about the items provided in the ItemMetadata document. See subclause 5.2.17.

Table 20 — Arguments for *GetRendererItemInfo()*

Argument	Direction	Related State Variable
ItemInfoFilter	*IN*	*A_ARG_TYPE_ItemInfoFilter*
ItemMetadataList	*IN*	*A_ARG_TYPE_Result*
ItemRenderingInfoList	*OUT*	*A_ARG_TYPE_RenderingInfoList*

5.4.7.2 Dependency on State

None.

5.4.7.3 Effect on State

None.

5.4.7.4 Errors

Table 21 — Error Codes for *GetRendererItemInfo()*

errorCode	errorDescription	Description
400-499	TBD	See Control clause in [14].
500-599	TBD	See Control clause in [14].
600-699	TBD	See Control clause in [14].

5.4.8 *GetFeatureList()*

This required action returns a *Features XML Document* describing which optional *ConnectionManager Features* this device supports, if any.

5.4.8.1 Arguments

Table 22 — Arguments for *GetFeatureList()*

Argument	Direction	Related State Variable
FeatureList	*OUT*	*FeatureList*

5.4.8.2 Dependency on State

None.

5.4.8.3 Effect on State

None.

5.4.8.4 Errors

Table 23 — Error Codes for *GetFeatureList()*

errorCode	errorDescription	Description
400-499	TBD	See Control clause in [14].
500-599	TBD	See Control clause in [14].
600-699	TBD	See Control clause in [14].

5.4.9 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 24 — Common Error Codes

errorCode	errorDescription	Description
400-499	TBD	See Control clause in [14].
500-599	TBD	See Control clause in [14].
600-699	TBD	See Control clause in [14].
701	Incompatible protocol info	The connection cannot be established because the protocol info argument is incompatible.
702	Incompatible directions	The connection cannot be established because the directions of the involved ConnectionManagers (source/sink) are incompatible.
703	Insufficient network resources	The connection cannot be established because there are insufficient network resources (bandwidth, channels, etc.).
704	Local restrictions	The connection cannot be established because of local restrictions in the device. This might happen, for example, when physical resources on the device are already in use by other connections.
705	Access denied	The connection cannot be established because the client is not permitted to access the specified ConnectionManager.
706	Invalid connection reference	The connection reference argument does not refer to a valid connection established by this service.
707	Not in network	The connection cannot be established because the ConnectionManagers are not part of the same physical network.
708	Connection Table overflow	The connection cannot be established because the specified ConnectionManager has instantiated the maximum number of simultaneous connections it has room for in its internal data structures. Closing one connection will resolve the issue.
709	Internal processing resources exceeded	The connection cannot be established because the device does not have sufficient internal processing resources to handle the new connection. Closing one or more connections on this device may resolve the issue.
710	Internal memory resources exceeded	The connection cannot be established because the device does not have sufficient internal memory resources to handle the new connection. Closing one or more connections on this device may resolve the issue.
711	Internal storage system capabilities exceeded	The connection cannot be established because the device does not have sufficient internal storage system capabilities to handle the new connection. Closing one or more connections on this device may resolve the issue.

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 Control clause in [14] for more details.

6 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>GetProtocolInfo</name>
         <argumentList>
            <argument>
               <name>Source</name>
```

```
            <direction>out</direction>
            <relatedStateVariable>
               SourceProtocolInfo
            </relatedStateVariable>
         </argument>
         <argument>
            <name>Sink</name>
            <direction>out</direction>
            <relatedStateVariable>
               SinkProtocolInfo
            </relatedStateVariable>
         </argument>
      </argumentList>
   </action>
   <action>
      <name>PrepareForConnection</name>
      <argumentList>
         <argument>
            <name>RemoteProtocolInfo</name>
            <direction>in</direction>
            <relatedStateVariable>
               A_ARG_TYPE_ProtocolInfo
            </relatedStateVariable>
         </argument>
         <argument>
            <name>PeerConnectionManager</name>
            <direction>in</direction>
            <relatedStateVariable>
               A_ARG_TYPE_ConnectionManager
            </relatedStateVariable>
         </argument>
         <argument>
            <name>PeerConnectionID</name>
            <direction>in</direction>
            <relatedStateVariable>
               A_ARG_TYPE_ConnectionID
            </relatedStateVariable>
         </argument>
         <argument>
            <name>Direction</name>
            <direction>in</direction>
            <relatedStateVariable>
               A_ARG_TYPE_Direction
            </relatedStateVariable>
         </argument>
         <argument>
            <name>ConnectionID</name>
            <direction>out</direction>
            <relatedStateVariable>
               A_ARG_TYPE_ConnectionID
            </relatedStateVariable>
         </argument>
         <argument>
            <name>AVTransportID</name>
            <direction>out</direction>
            <relatedStateVariable>
               A_ARG_TYPE_AVTransportID
            </relatedStateVariable>
         </argument>
         <argument>
            <name>RcsID</name>
            <direction>out</direction>
            <relatedStateVariable>
               A_ARG_TYPE_RcsID
            </relatedStateVariable>
         </argument>
      </argumentList>
   </action>
```

```
<action>
   <name>ConnectionComplete</name>
   <argumentList>
      <argument>
         <name>ConnectionID</name>
         <direction>in</direction>
         <relatedStateVariable>
            A_ARG_TYPE_ConnectionID
         </relatedStateVariable>
      </argument>
   </argumentList>
</action>
<action>
   <name>GetCurrentConnectionIDs</name>
   <argumentList>
      <argument>
         <name>ConnectionIDs</name>
         <direction>out</direction>
         <relatedStateVariable>
            CurrentConnectionIDs
         </relatedStateVariable>
      </argument>
   </argumentList>
</action>
<action>
   <name>GetCurrentConnectionInfo</name>
   <argumentList>
      <argument>
         <name>ConnectionID</name>
         <direction>in</direction>
         <relatedStateVariable>
            A_ARG_TYPE_ConnectionID
         </relatedStateVariable>
      </argument>
      <argument>
         <name>RcsID</name>
         <direction>out</direction>
         <relatedStateVariable>
            A_ARG_TYPE_RcsID
         </relatedStateVariable>
      </argument>
      <argument>
         <name>AVTransportID</name>
         <direction>out</direction>
         <relatedStateVariable>
            A_ARG_TYPE_AVTransportID
         </relatedStateVariable>
      </argument>
      <argument>
         <name>ProtocolInfo</name>
         <direction>out</direction>
         <relatedStateVariable>
            A_ARG_TYPE_ProtocolInfo
         </relatedStateVariable>
      </argument>
      <argument>
         <name>PeerConnectionManager</name>
         <direction>out</direction>
         <relatedStateVariable>
            A_ARG_TYPE_ConnectionManager
         </relatedStateVariable>
      </argument>
      <argument>
         <name>PeerConnectionID</name>
         <direction>out</direction>
         <relatedStateVariable>
            A_ARG_TYPE_ConnectionID
         </relatedStateVariable>
```

```
            </argument>
            <argument>
               <name>Direction</name>
               <direction>out</direction>
               <relatedStateVariable>
                  A_ARG_TYPE_Direction
               </relatedStateVariable>
            </argument>
            <argument>
               <name>Status</name>
               <direction>out</direction>
               <relatedStateVariable>
                  A_ARG_TYPE_ConnectionStatus
               </relatedStateVariable>
            </argument>
         </argumentList>
      </action>
      <action>
         <name>GetRendererItemInfo</name>
         <argumentList>
            <argument>
               <name>ItemInfoFilter</name>
               <direction>in</direction>
               <relatedStateVariable>
                  A_ARG_TYPE_ItemInfoFilter
               </relatedStateVariable>
            </argument>
            <argument>
               <name>ItemMetadataList</name>
               <direction>in</direction>
               <relatedStateVariable>
                  A_ARG_TYPE_Result
               </relatedStateVariable>
            </argument>
            <argument>
               <name>ItemRenderingInfoList</name>
               <direction>out</direction>
               <relatedStateVariable>
                  A_ARG_TYPE_RenderingInfoList
               </relatedStateVariable>
            </argument>
         </argumentList>
      </action>
      <action>
         <name>GetFeatureList</name>
         <argumentList>
            <argument>
               <name>FeatureList</name>
               <direction>out</direction>
               <relatedStateVariable>
                  FeatureList
               </relatedStateVariable>
            </argument>
         </argumentList>
      </action>
   </actionList>
   <serviceStateTable>
      <stateVariable sendEvents="yes">
         <name>SourceProtocolInfo</name>
         <dataType>string</dataType>
      </stateVariable>
      <stateVariable sendEvents="yes">
         <name>SinkProtocolInfo</name>
         <dataType>string</dataType>
      </stateVariable>
      <stateVariable sendEvents="yes">
         <name>CurrentConnectionIDs</name>
         <dataType>string</dataType>
```

```
      </stateVariable>
      <stateVariable sendEvents="no">
         <name>A_ARG_TYPE_ConnectionStatus</name>
         <dataType>string</dataType>
         <allowedValueList>
            <allowedValue>OK</allowedValue>
            <allowedValue>ContentFormatMismatch</allowedValue>
            <allowedValue>InsufficientBandwidth</allowedValue>
            <allowedValue>UnreliableChannel</allowedValue>
            <allowedValue>Unknown</allowedValue>
         </allowedValueList>
      </stateVariable>
      <stateVariable sendEvents="no">
         <name>A_ARG_TYPE_ConnectionManager</name>
         <dataType>string</dataType>
      </stateVariable>
      <stateVariable sendEvents="no">
         <name>A_ARG_TYPE_Direction</name>
         <dataType>string</dataType>
         <allowedValueList>
            <allowedValue>Input</allowedValue>
            <allowedValue>Output</allowedValue>
         </allowedValueList>
      </stateVariable>
      <stateVariable sendEvents="no">
         <name>A_ARG_TYPE_ProtocolInfo</name>
         <dataType>string</dataType>
      </stateVariable>
      <stateVariable sendEvents="no">
         <name>A_ARG_TYPE_ConnectionID</name>
         <dataType>i4</dataType>
      </stateVariable>
      <stateVariable sendEvents="no">
         <name>A_ARG_TYPE_AVTransportID</name>
         <dataType>i4</dataType>
      </stateVariable>
      <stateVariable sendEvents="no">
         <name>A_ARG_TYPE_RcsID</name>
         <dataType>i4</dataType>
      </stateVariable>
      <stateVariable sendEvents="no">
         <name>A_ARG_TYPE_ItemInfoFilter</name>
         <dataType>string</dataType>
      </stateVariable>
      <stateVariable sendEvents="no">
         <name>A_ARG_TYPE_Result</name>
         <dataType>string</dataType>
      </stateVariable>
      <stateVariable sendEvents="no">
         <name>A_ARG_TYPE_RenderingInfoList</name>
         <dataType>string</dataType>
      </stateVariable>

      <stateVariable sendEvents="no">
         <name>ClockUpdateID</name>
         <dataType>ui4</name>
      </stateVariable>
      <stateVariable sendEvents="yes">
         <name>DeviceClockInfoUpdates</name>
         <dataType>string</dataType>
      </stateteVariable>
   </serviceStateTable>
</scpd>
```

7 Test

No semantics tests have been defined for this service.

Annex A
(normative)

Protocol Specifics

A.1 Application to HTTP Streaming

A.1.1 *ProtocolInfo* Definition

Streaming data via the HTTP GET method is defined by the Internet standard Request For Comment document entitled HyperText Transport Protocol – HTTP/1.1 [38]. While it is certainly possible to use other HTTP methods such as PUT or POST, this document focuses on the HTTP GET method. The <protocol> part of *ProtocolInfo* shall be "`http-get`". The <network> part of *ProtocolInfo* is not used for the HTTP case since all devices belong to the same network. An asterisk ("*") is used instead. The <contentFormat> part for HTTP GET is described by a MIME type [39].

An example of protocol information for HTTP GET, in this case referring to an audio file, is:

```
http-get:*:audio/mpeg:*
```

A.1.2 Implementation of *PrepareForConnection()*

In addition to any non-protocol related preparation tasks such as the one described in Annex C.5, a device's *PrepareForConnection()* implementation may also perform some preparation tasks that are related to the protocol that is about to be used to transfer the content. However, since the HTTP GET connection is initiated and maintained by the sink device, the source device typically does not need to perform any protocol-related preparation tasks because HTTP GET requests are handled by the device's underlying http-server. Therefore, if a MediaServer does not need to perform any non-protocol-related preparation tasks, it will (in most cases) not need to implement *PrepareForConnection()*. Although not required, the MediaRenderer device (the receiving end of the HTTP stream), may choose to pre-allocate a TCP/IP socket in order to ensure that this resource is available when the content transfer is initiated; that is: when *AVTransport::SetAVTransportURI()* is invoked.

A.1.3 Implementation of *ConnectionComplete()*

In addition to the non-protocol related cleanup tasks such as those described in Annex C.5, a device's *ConnectionComplete()* implementation may also perform some cleanup tasks that are related to the protocol that was used to transfer the content. The cleanup tasks that a device performs depend directly on the implementation of *PrepareForConnection()*. In general, when using the HTTP GET protocol, a MediaServer does not have any protocol-related cleanup tasks to perform because the MediaServer's *PrepareForConnection()* typically does not perform any protocol-related preparation. On a MediaRenderer device, *ConnectionComplete()* shall release any protocol-related resources that were allocated during *PrepareForConnection()*. For example, if a MediaRenderer chooses to pre-allocate a TCP/IP socket during *PrepareForConnection()*, the device's *ConnectionComplete()* action shall release the socket associated with that connection.

A.1.4 Automatic Connection Cleanup

Since control points may establish connections, and then leave the UPnP network forever, protocols supported by the ConnectionManager need to have a built-in automatic mechanism to cleanup stale connections. For HTTP connections, automatic cleanup should be performed by the AVTransport instance.

On the UPnP level, this will appear as an (evented) change in state variable *CurrentConnectionIDs*.

A.2 Application to RTSP/RTP/UDP Streaming

A.2.1 *ProtocolInfo* Definition

Streaming data via RTSP is defined by the Internet standard Request For Comment document entitled Real Time Streaming Protocol. (http://www.ietf.org/rfc/rfc2326.txt). The actual Audio/Video data packets are sent out-of-band with respect to RTSP. RTSP does not need a particular protocol for this. Since usually RTP (http://www.ietf.org/rfc/rfc1889.txt) over UDP is used, the protocol for RTSP-based streams is defined as RTSP/RTP/UDP. This ensures that two ConnectionManagers that can send and receive RTSP also send and receive using the same Audio/Video data Connection protocol. The <protocol> part of *ProtocolInfo* shall be "`rtsp-rtp-udp`". The <network> part of *ProtocolInfo* is not used for the RTSP/RTP/UDP case since all devices belong to the same network. An asterisk ("*") is used instead. RTP packets contain a standardized 7-bit payload type identifier, see http://www.iana.org/assignments/rtp-parameters or http://www.ietf.org/rfc/rfc1890.txt. Each payload type has a unique encoding name. This payload type name is used as the <contentFormat> part of the *ProtocolInfo* string.

An example of protocol information for RTSP/RTP/UDP with MPEG video payload is:

```
rtsp-rtp-udp:*:MPV:*
```

A.2.2 Implementation of *PrepareForConnection()*

In addition to the non-protocol related preparation task such as those described in Annex C.5, a device's *PrepareForConnection()* implementation may also perform some preparation tasks that are related to the protocol that is about to be used to transfer the content. However, since RTSP/RTP/UDP sessions are initiated and maintained by the sink device, the source device typically does not need to perform any protocol-related preparation tasks. Therefore, if a MediaServer does not need to perform any non-protocol-related preparation tasks, it will (in most cases) not need to implement *PrepareForConnection()*. Although not required, the MediaRenderer device (the receiving end of the RTP/RTP/UDP stream) may choose to pre-allocate an RTSP/RTP/UDP connection in order to ensure that this resource is available when the content transfer is initiated; that is: when *AVTransport::SetAVTransportURI()* is invoked.

A.2.3 Implementation of *ConnectionComplete()*

In addition to the non-protocol related cleanup tasks such as those described in Annex C.5, a device's *ConnectionComplete()* implementation may also perform some cleanup tasks that are related to the protocol that was used to transfer the content. The cleanup tasks that a device performs depend directly on the implementation of *PrepareForConnection()*. In general, when using the RTSP/RTP/UDP protocol, a MediaServer does not have any protocol-related cleanup tasks to perform because the MediaServer's *PrepareForConnection()* typically does not perform any protocol-related preparation. On a MediaRenderer device, *ConnectionComplete()* shall release any protocol-related resources that were allocated during *PrepareForConnection()*. For example, if a MediaRenderer chooses to pre-allocate a RTSP/RTP/UDP connection during *PrepareForConnection()*, the device's *ConnectionComplete()* shall release that connection.

A.2.4 Automatic Connection Cleanup

Since control points may establish connections, and then leave the UPnP network forever, protocols supported by the ConnectionManager need to have a built-in automatic mechanism to cleanup stale connections. For RTSP connections, automatic cleanup should be performed by the AVTransport instance.

On the UPnP level, this will appear as an (evented) change in state variable *CurrentConnectionIDs*.

A.3 Application to Device-Internal Streaming

For the purpose of this service definition an INTERNAL protocol is defined for use over internal connections. An internal connection is defined as a connection within a single device. An example of such a connection is between a Tuner subsystem and a Display subsystem in a conventional TV. Since this connection is internal to the device, no streaming data will flow on

the UPnP network, and the actual content-format used inside the device can be proprietary. The resulting *ProtocolInfo* and content-URI that need to be defined for these types of connections can therefore be very simple.

An internal connection shall use the INTERNAL protocol. For this protocol, the <protocol> part of *ProtocolInfo* shall be "*internal*". Within this protocol scope the <network> part of *ProtocolInfo* is defined as the device's IP-address, as a string, in the well-known dotted decimal notation. The <contentFormat> part of *ProtocolInfo* is proprietary.

An example of protocol information for INTERNAL is:

```
internal:161.88.59.212:mpeg2:to-local-display
```

The implementation of the *PrepareForConnection()* and *ConnectionComplete()* actions for this protocol type is proprietary (vendor specific).

A.4 Application to IEC61883 Streaming

A.4.1 *ProtocolInfo* Definition

The basis for real time data transmission on the IEEE1394 bus using the IEC61883 protocol is the Common Isochronous Packet (CIP) which consists of a CIP header and data blocks embedded in an IEEE1394 compliant isochronous packet. The <protocol> part of *ProtocolInfo* shall be "`iec61883`".

The <network> part of *ProtocolInfo* for the IEC61883 protocol uniquely identifies the set of connected IEEE1394 devices on a specific bus segment. It is defined as a bin.hex encoding of the GUID (Globally Unique ID) of the 1394 Isochronous Resource Manager node. This identification is not persistent, and will, in general, change when 1394 devices are added to or removed from the 1394 network. These changes will lead to changes in the *SourceProtocolInfo* and *SinkProtocolInfo* state variables, and, through eventing, interested control points will be notified of the new streaming possibilities of the new 1394 network segmentation.

The stream types include all content formats supported by the family of IEC61883 Standards. These formats are uniquely identified by the FMT and FDF values in the CIP header, The following table lists the formats supported by the IEC61883-2 to 5 International Standards and by IEC61883-6 PAS (Publicly Available Specification; *that is: not yet fulfilling all requirements for a standard*).

Table A.1 — <contentFormat> for Protocol IEC61883

<contentFormat> for Protocol IEC61883	IEC Version	Description
"UNKNOWN_STREAM"		
"DVCR_STD_DEF_525_60"	IEC61883-2	525-60 system: the 525-line system with a frame frequency of 29.97 Hz
"DVCR_STD_DEF_625_50"	IEC61883-2	625-50 system: the 625-line system with a frame frequency of 25.00 Hz
"DVCR_STD_DEF_HI_COMPRESS_525_60"	IEC61883-5	SDL525-60 system: The standard definition for high compression mode 525-line system with a frame frequency of 29.97 Hz.
"DVCR_STD_DEF_HI_COMPRESS_625_50"	IEC61883-5	SDL625-50 system: The standard definition for high compression mode 625-line system with a frame frequency of 25.00 Hz.
"DVCR_HI_DEF_1125_60"	IEC61883-3	1125-60 system: the 1125-line system with a frame frequency of 30.00 Hz
"DVCR_HI_DEF_1250_50"	IEC61883-3	1250-50 system: the 1250-line system with a frame frequency of 25.00 Hz
"SMPTE_D7_525_60"	IEC61883-2	SMPTE-D7 525-60 system
"SMPTE_D7_625_50"	IEC61883-2	SMPTE-D7 625-50 system
"MPEG2_TS"	IEC61883-4	MPEG2-TS
"AUDIO_MUSIC_8_24_IEC_60958"	IEC61883-6	IEC 60958 conformant
"AUDIO_MUSIC_8_24_RAW_AUDIO"	IEC61883-6	Raw audio
"AUDIO_MUSIC_8_24_MIDI"	IEC61883-6	MIDI conformant

IEC61883 connections are set up between iPCRs (input Plug Control Registers) and oPCRs (output Plug Control Registers). A content item is connected through an oPCR to one or more iPCRs on a different device. An IEC61883 device can have zero or more iPCRs and oPCRs.

The <additionalInfo> field identifies the PCR in the IEC61883 network, and is defined as follows:

```
<GUID>;<PCR-index>
```

where

- <GUID> = bin.hex encoding of the device's node_vendor_id and chip_id (2 quadlets, together also referred to as GUID)
- <PCR-index> = zero-based integer index identifying the plug within the device

An example of protocol information for IEC61883 is:

```
iec61883:0000f00200001114:MPEG2_TS:00ba0091c9231222;0
```

A.4.2 Implementation of *PrepareForConnection()*

In addition to the non-protocol related preparation task such as those described in Annex C.5, a device's *PrepareForConnection()* implementation may also perform some preparation tasks that are related to the protocol that is about to be used to transfer the content Although IEEE1394/IEC61883 is an allocation-based protocol, the source device is not required to perform any protocol related preparation. It is the sink device that is responsible for allocating the underlying IEEE1394/IEC61883 connection.

In order to manage isochronous data transmissions, IEC61883 defines the concept of plug and specialized registers called MPR (Master Plug Register) and PCR (Plug Control Register). These registers are used to initiate and stop transmissions. The set of procedures to control the real time data flow by manipulating the PCRs is called CMP (Connection Management Procedures). Data transmission between devices is possible when an output plug on the source

device is connected to an input plug on the sink device via an isochronous channel. The data flow from a source device is controlled by the oMPR (output Master Plug Register) of the device and one oPCR (output PCR). Similarly, the data flow to a sink device is controlled by the iMPR (input MPR) and one iPCR (input PCR). The address map for these registers is well defined in conformance with ISO/IEC13213 (ANSI/IEEE1212). Devices can modify PCR values of remote nodes using asynchronous transactions.

Using the information in the *PrepareForConnection()* input arguments, the sink device locates the IEEE1394 address of the source device (its GUID is part of the <additionalInfo> field of the *ProtocolInfo* string), and program the appropriate oPCR register to initiate the streaming. The sink device is free to choose any of its own iPCRs. The sink device shall follow the exact procedure defined by IEC61883, which includes the allocation of IEEE1394 bandwidth and an IEEE1394 channel. Upon subsequent IEEE1394 bus resets, the *sink* device (the device that established the connection) shall try to restore any existing connections that it has established.

If the *ProtocolInfo* references an oPCR that is already in use, two situations occur:

- The same content-format is already being streamed via the oPCR. In this case, the sink device performs an IEC61883 *overlay* connection.
- A different content-format is already being streamed via the oPCR. In this case, the sink device will return an error.

IEC61883 broadcast-in and broadcast-out connections are not supported by the ConnectionManager.

A.4.3 Implementation of *ConnectionComplete()*

In addition to the non-protocol related cleanup tasks such as those described in Annex C.5, a device's *ConnectionComplete()* implementation may also perform some cleanup tasks that are related to the protocol that was used to transfer the content. The cleanup tasks that a device performs depend directly on the implementation of *PrepareForConnection()*. In general, when using the IEEE1394/IEC61883 protocol, the source device does not have any protocol-related cleanup tasks to perform because the device's *PrepareForConnection()* typically does not perform any protocol-related preparation. On a sink device, *ConnectionComplete()* shall release the IEEE1394/IEC61883 connection that was allocated during *PrepareForConnection()*. The sink device shall follow the IEC61883 procedure for releasing the channel which includes:

- Modifying the corresponding fields of the source oPCR and sink iPCR according to CMP procedures.
- Deallocate the IEEE1394 resources. If the oPCR becomes unconnected (that is: this is the last IEC61883 connection on that IEEE1394 channel), the IEEE1394 bandwidth and channel shall also be released.

Note: IEC61883 broadcast-in and broadcast-out connections are not supported by the ConnectionManager.

A.4.4 Automatic Connection Cleanup

Since control points may establish connections, and then leave the UPnP network forever, protocols supported by the ConnectionManager need to have a built-in automatic mechanism to cleanup stale connections. For the IEC61883 protocol, an established connection will continue forever, until there is a so-called *bus reset*. A bus reset will occur when there is a change in the physical network topology, for example, the network is split, joined with another network, or a device goes offline. After a bus reset, all IEEE1394 resources are released, and all devices that established IEC61883 connections have 1 second to re-establish them. Hence, the ConnectionManager on the sink device needs to check after a bus reset whether the source device is still on the network, and if not, cleanup any internal state referring to this connection. On the UPnP level, this will appear as an (evented) change in state variable *CurrentConnectionIDs*.

A.5 Application to Vendor-specific Streaming

To allow vendors to use their vendor-specific streaming protocols in a UPnP network in a controlled way, the ConnectionManager defines the generic protocol VENDOR for such protocols. The idea is to make the <protocol> part of *ProtocolInfo* unique, by requiring the use of the vendor's registered ICANN (Internet) domain name (similar to its use in vendor-specific UPnP service- and device-types). The remaining fields of the *ProtocolInfo* string (<network>, <contentFormat> and <additionalInfo>) are all vendor-specific, and may be wildcards ("*").

An example of a VENDOR protocol information is:

```
company.com:*:company-format-A:optional-setup-info
```

The implementation of the *PrepareForConnection()* and *ConnectionComplete()* actions for this protocol type is proprietary (vendor-specific).

Annex B
(normative)

CM features

B.1 Introduction

Annex B defines a set of extended functionalities for the ConnectionManager service, called *CM features*. These features have additional requirements beyond the general ConnectionManager service mechanisms to ensure interoperability. The requirements are defined on a per *CM feature* basis. When an implementation supports a specific *CM feature*, it shall support that feature according to these requirements.

Each *CM feature* shall have an integer version number. Later versions – indicated by a larger version number – shall support the full functionalitiy of all earlier, lower-numbered versions in the same way as the earlier version (that is, shall be backward compatible).

The names and versions for each implementation-supported feature are returned by the *GetFeatureList()* action. The format of the returned information is defined by [11].

The normative names for the *CM feature*s are listed in Table B.1. All *CM feature*s are allowed. A vendor may use vendor defined feature names. In this case, the vendor-defined *CM feature* name shall be prefixed with the vendor's ICANN domain name followed by the underscore "_" and any vendor defined value shall be prefixed with "X_".

Example: company.com_MyFeature.

Table B.1 — *CM features*

Name	Description
CLOCKSYNC	Synchronized Playback Support. See Annex B.2.
Vendor-defined	

B.2 Requirements for the *CLOCKSYNC feature*, Version 1

The *CLOCKSYNC feature* is defined by this specification for the purposes of describing devices that enable synchronized playback.

The ConnectionManager service that supports the *CLOCKSYNC feature* provides clock synchronization guide information. It shall satisfy the following requirements.

Support for the *CLOCKSYNC feature* shall be indicated by including the following *Features XML fragment* in the *Features XML Document* value of the *FeatureList* state variable. The data is encapsulated within zero or more `<deviceClockInfo>` elements.

The following example shows a "template" for the format of the *CLOCKSYNC feature*. Additional elements and/or attributes may be added to future versions of this specification. Furthermore, a 3rd-party vendor may add vendor-defined elements or attributes. Even if vendors define new metadata, the values for <Feature> attributes shall match the values described in Table B.2.

In order to eliminate element or attribute naming conflicts, the name of any vendor-defined element or attribute shall follow the rules set forth in subclause 4.4.3. All control points are expected to gracefully ignore any element or attribute that it does not understand.

The following XML template includes the *vendor* character style which shows the fields that need to be filled out by individual implementations.

```
<Feature name="CLOCKSYNC" version="1">
   <deviceClockInfo
    id="unique ID for this deviceClock instance"
```

```
    updateID="ClockUpdateID for this device">
      <syncProtocolID>Id of clock sync protocol used</syncProtocolID>
      <masterClockID>Id of the master clock</masterClockID>
      <accuracy>max deviation from master clock (nano-sec)</accuracy>
      <supportedTimestamps
       id="unique id of this instance"
       protocol="transfer protocol"
       format="MIME-type">
         Id of timestamp mechanism used with this protocol/format
      </supportedTimestamps>
      <!-- Other supportedTimestamps entries go here -->
      <!-- Other vendor-defined metadata goes here -->
   </deviceClockInfo>
</Feature>
```

The *CLOCKSYNC feature* `<Feature>` element has the following required characteristics:

Table B.2 — Required characteristics of the *CLOCKSYNC feature* element

Name	R/A	XML Form	Type	Description
version	R	Attribute of <Feature>	xsd:unsigndInt	Indicates the *CLOCKSYNC feature* version. Shall be set to "1" for this version.
deviceClockInfo	CR a	Child element of <Feature>	xsd:string	Indicates that the device has synchronized its local time-of-day clock to some master clock on the network. Includes information about the synchronization protocol that was used. Contains exactly one instance of each of the following elements except for the <supportedTimestamps> element which may occur zero or more times. Vendor-defined attributes may be used with the <deviceClockInfo> element. Vendor-defined elements may appear as children of the <deviceClockInfo> element.
id	R	Attribute of <deviceClockInfo>	xsd:string	A unique ID for this instance of the <deviceClockInfo> element. The specific value is not important but it shall be unique within the scope of the *CLOCKSYNC feature*. Additionally, since the value is referenced by other data structures (for example, the *upnp:resExt::clockSync* property within the ContentDirectory service), the value shall be persisted across reboots.
updateID	R	Attribute of <deviceClockInfo>	xsd:unsignedInt	Contains the value of the *ClockUpdateID* state variable that resulted when the <deviceClockInfo> element was added or modified.
syncProtocolID	R	Child element of < deviceClockInfo >	xsd:string	Shall appear exactly once. Identifies the clock synchronization protocol that was used to synchronize the implementation's local time-of-day clock. Contains one of the following values listed in Table B.3. Vendor-defined attributes may be used with the <syncProtocolID> element. Vendor-defined elements are not allowed as children of the <syncProtocolID> element.
masterClockID	R	Child element of < deviceClockInfo >	xsd:string	Shall appear exactly once. Identifies the master clock to which this implementation has synchronized its local time-of-day clock. Value shall conform to the format listed in Table B.4 for the clock synchronization protocol identified by the <syncProtocolID> element. Vendor-defined attributes may be used with the <masterClockID> element. Vendor-defined elements are not allowed as children of the <masterClockID> element.
accuracy	R	Child element of < deviceClockInfo >	xsd:unsignedInt	Shall appear exactly once. Indicates the maximum number of nano-seconds that this implementation's local clock might deviate from the Master Clock. For example, a value of 1000 indicated that the implementation's local clock might deviate up to one micro-second (1000 nano-seconds) from the Master Clock. Typically, this is characteristic of the synchronization protocol, network topology, and the device itself. Vendor-defined attributes may be used with the <accuracy> element. Vendor-defined elements are not allowed as children of the <accuracy> element.

Name	R/A	XML Form	Type	Description
supportedTimestamps	A	Child element of <deviceClockInfo>	xsd:string	Shall appear zero or more times. Identifies the timestamp mechanism supported by the device when content is streamed to/from the device using the transfer protocol and media format indicated by the protocol and format attributes (see below). The list of allowed values is defined in Table B.5. It contains the attributes and elements listed after the table. Vendor-defined attributes may be used with the <supportedTimestamps> element.
id	R	Attribute of <supportedTimestamps>	xsd:string	Contains a unique ID for this instance of the <supportedTimestamps> element. Format and value are vendor-defined but the value shall be unique within this instance of the <deviceClockInfo> element. Additionally, since this value is referenced by other data structures (for example, the upnp:resExt::clockSync property within the ContentDirectory service), the value shall be persisted across reboots.
protocol	R	Attribute of <supportedTimestamps>	xsd:string	Identifies the transfer protocol that is associated with the timestamp mechanism listed in this <supportedTimestamps> element. The value of this attribute shall be one of the values listed in the "protocol" column of Table C.1 in Annex C.2.
format	R	Attribute of <supportedTimestamps>	xsd:string	Identifies the media format MIME-type associated with the timestamp mechanism listed in this <supportedTimestamps> element. The value of this attribute shall comply with the "content-format" column of Table C.1 in Annex C.2. This includes a value of "*" which means that the specified timestamp mechanism (<supportedTimestamps>) is supported with any of the media formats that are supported by the device when using the indicated protocol attribute.
a conditionally required when describing the *CLOCKSYNC feature* and appears zero or more times.				

Table B.3 — Allowed values for the <syncProtocolID> element

Clock Synchronization Protocol ID	Description
802.1AS	IEEE-802.1AS. See [40], [41] and [42] for details.
NTP	NetworkTime Protocol time synchronization mechanism. See [43] for details.
SNTP	Simple NTP time synchronization. See [44] for details.
Vendor-defined	A value defined by the vendor.

Table B.4 — Allowed formats for the <masterClockID> element.

Clock Synchronization Protocol ID	Format of the <masterClockID> value
802.1AS	8-byte binary sequence with the following format: <High 24-bits MAC> 0xFF 0xFE <Low 24-bits MAC> Where <High 24-bits MAC> = The top 24 (most significant) bits of the device's 48-bits MAC address. <Low 24-bits MAC> = The bottom 24 (least significant) bits of the device's 48-bit MAC address. Note: The most significant byte of the MAC address is stored in the first (lowest addressed) byte of the sequence.
NTP	URL of the time server Note: see www.ntp.org
SNTP	URL of the time server Note: see www.ntp.org
Vendor-defined	A value defined by the vendor.

Table B.5 — Allowed values for the <supportedTimestamps> element.

Timestamp Mechanism	Description
RTP+IEEE-1733	Packets are timestamped according to the IEEE-1733 specification. See [45] for details.
Identity	This is a pre-defined mapping that states that the frequencies of the Media Clock and the Master Clock are identical. This is the default Time Stamp Mechanism and it can be used with both HTTP and RTP. When Identity is used with RTP, "Media Clock" refers to the RTP wallclock, as defined in [23], [24], and [46].
Vendor-defined	A value defined by the vendor.

Example:

The following example shows that this implementation does the following:

- Advertised three clock synchronization mechanisms, one for each <deviceClockInfo> element.

1) The first clock synchronization mechanism has synchronized its local clock using IEEE 802.1AS (<syncProtocolID>) protocol with a clock master whose ID is 123456FFFE789ABC (<masterClockID>). This device will remain within 10ns (<accuracy>) of the master clock. This device is able to use this synchronization mechanism with any content transported with RTP, whose content is timestamped according to [45] (<supportedTimestamps>).

4) The second clock synchronization mechanism has synchronized its local clock using IEEE 802.1AS (<syncProtocolID>) protocol with a clock master whose ID is 123456FFFE789ABC (<masterClockID>). This device will remain within 10ns (<accuracy>) of the master clock. This device is able to use this synchronization mechanism with any content transported with RTP or HTTP through the use of Identity timestamp mechanism. Identity is a pre-defined mapping that states that the frequencies of the Media Clock and the Master Clock are identical. This is the default Time Stamp Mechanism and it can be used with both HTTP and RTP. When Identity is used with RTP, "Media Clock" refers to the RTP wallclock according to [23], [24], and [46].

5) The third clock synchronization mechanism has synchronized its local clock using IEEE 802.1AS (<syncProtocolID>) protocol with a different clock master whose ID is 789ABCFFFE123686 (<masterClockID>). This device will remain within 100ns (<accuracy>) of the master clock. This device is able to use this synchronization

mechanism with WMA or MP3 audio content transported with RTP, so long as the content is timestamped according to [45].

```
<Features
  xmlns="urn:schemas-upnp-org:av:cm-featureList"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="
      urn:schemas-upnp-org:av:cm-featureList
      http://www.upnp.org/schemas/av/cm-featureList.xsd">
  <Feature name="CLOCKSYNC" version="1">
      <!-- Clock info for a local device using RTP+IEEE-1733 mechanism -->
      <deviceClockInfo id="Clock#1" updateID="1">
          <syncProtocolID>802.1AS</syncProtocolID>
          <masterClockID>123456FFFE789ABC</masterClockID>
          <accuracy>10</accuracy>
          <!-- RTP+IEEE-1733 timestamps for RTP with all media formats -->
          <supportedTimestamps id="rtp_all" protocol="rtsp-rtp-udp" format="*">
           RTP+IEEE-1733
          </supportedTimestamps>
      </deviceClockInfo>
      <!-- Clock info for a local device using IDENTITY mechanism -->
      <deviceClockInfo id="Clock#2" updateID="1">
          <syncProtocolID>802.1AS</syncProtocolID>
          <masterClockID>123456FFFE789ABC</masterClockID>
          <accuracy>10</accuracy>
          <!-- IDENTITY timestamps for RTP/HTTP with all media formats -->
          <supportedTimestamps id="Timestamp-1" protocol="rtsp-rtp-udp" format="*">
            Identity
          </supportedTimestamps>
          <supportedTimestamps id="Timestamp-2" protocol="http-get" format="*">
            Identity
          </supportedTimestamps>
      </deviceClockInfo>
      <!-- Clock info for a different clock master using RTP+IEEE-1733 -->
      <deviceClockInfo id="Clock#3" updateID="3">
          <syncProtocolID>802.1AS</syncProtocolID>
          <masterClockID>789ABCFFFE123686</masterClockID>
          <accuracy>100</accuracy>
          <!-- RTP+IEEE-1733 timestamps for RTP but only with wma & mp3 -->
        <supportedTimestamps id="rtp_wma" protocol="rtsp-rtp-udp" format="audio/wma">
            RTP+IEEE-1733
          </supportedTimestamps>
          <supportedTimestamps id="rtp_mp3" protocol="rtsp-rtp-udp" format="audio/mp3">
            RTP+IEEE-1733
          </supportedTimestamps>
      </deviceClockInfo>
  </Feature>
</Features>
```

Annex C
(informative)

Theory of Operation

C.1 Purpose

The purpose of the ConnectionManager is to enable control points to:

a) perform capability matching between source/server devices and sink/renderer devices. This involves both:
   1) content-format matching (for example, mp3 – mp3)
   2) transport (streaming) protocol matching (for example, http – http)
b) find information about currently ongoing streams in the network, for example:
   1) find the source device sending content to a given renderer device
   2) find the renderer devices served by a given source device or content resource
   3) find all streams going on in the network
c) setup and teardown connections between devices (when required by the streaming protocol)

C.2 *ProtocolInfo* Concept

While the UPnP Architecture describes, and prescribes, many aspects of devices that are required for a certain level of interoperability, it does not describe anything related to streaming between devices. The purpose of the ConnectionManager service is to make these aspects of devices explicit, so that control points are able to make intelligent choices, present intelligent user interfaces, and initiate (and terminate) streams between controlled devices via UPnP actions. UPnP-defined protocols are used to initiate (and terminate) the stream, even though they are not used to stream the actual data *packets*.

The ConnectionManager service defines the notion of *ProtocolInfo* as information needed by a control point in order to determine (a certain level of) compatibility between the streaming mechanisms of two UPnP controlled devices. For example, it contains the transport protocols supported by a device, for input or output, as well as other information such as the content formats (encodings) that can be sent, or received, via the transport protocols. Note that, while UPnP prescribes the use of HTTP for controlling devices via SOAP, it does not require HTTP to be used for all kinds of (Audio and Video) streaming in a UPnP network.

In the context of this document, the term *ProtocolInfo* is used to describe a string formatted as:

<protocol>":"<network>":"<contentFormat>":"<additionalInfo>

where each of the 4 elements may be a wildcard "*". Control points can match *ProtocolInfo* by (protocol-independent) string comparison operations on the <protocol>, <network> and <contentFormat> elements, taking into account the "*" wildcard, which matches with anything. It is recommended that control points perform string matching using case-insensitive comparison. However, devices are required to provide the <protocol>, <network>, and <contentFormat> strings exactly as prescribed by this and other specifications.

When performing protocol matching, control points have basically three different sources for protocol information:

- The value of the *res@protocolInfo* property of the content item to be played, which is exposed by the ContentDirectory service.
- The Comma Separated Value list maintained in the *SinkProtocolInfo* state variable of the MediaRenderer device.
- The Comma Separated Value list maintained in the *SourceProtocolInfo* state variable of the MediaServer device.

Control points should match the content item's *res@protocolInfo* property value to one of the *ProtocolInfo* entries in the MediaRenderer's *SinkProtocolInfo* CSV list. In addition, a control point may want to check whether the content item's *res@protocolInfo* property value matches one of the entries in the MediaServer's *SourceProtocolInfo* CSV list to ensure that the MediaServer is currently capable of serving this content item.

The <additionalInfo> part does not need to match between source and sink. Its purpose is to convey any additional information needed to set up the out of band stream (for example, 1394 addresses). The structure of this 4th field is described later in this

The following table summarizes how the protocol info strings are defined for the protocols currently standardized by the ConnectionManager service, as well as for vendor-defined protocols. Annex A provides a more detailed explanation per protocol.

Table C.1 — Defined Protocols and their associated *ProtocolInfo* Values

Protocol Name	protocol	network	contentFormat	additionalInfo	Ref.
HTTP GET	"*http-get*"	"*" a	MIME-type.	Vendor-defined, may be "*".	A.1
RTSP/RTP/UDP	"*rtsp-rtp-udp*"	"*" b	Name of RTP payload type.	Vendor-defined, may be "*".	A.2
INTERNAL	"*internal*"	IP address of the device hosting the Connection-Manager.	Vendor-defined, may be "*".	Vendor-defined, may be "*".	A.3
IEC61883_EX1	"*iec61883_ex1*"	GUID of the 1394 bus Isochronous Resource Manager.	Name standardized by IEC61883.	upnp.org_GUID = <GUID-value>;<PCR-index>. See definitions below.	A.4
IEC61883	"*iec61883*"	GUID of the 1394 bus Isochronous Resource Manager.	Name standardized by IEC61883.	GUID and PCR index of the 1394 device. See IEC61883 exception below.	A.4
VENDOR	Registered ICANN domain name of vendor	Vendor-defined, may be "*".	Vendor-defined, may be "*".	Vendor-defined, may be "*".	A.5

a Since all devices supporting HTTP GET belong to the same IP network, the network does not need to be specified.

b Since all devices supporting RTSP/RTP/UDP belong to the same IP network, the network does not need to be specified.

C.2.1 4th Field – <additionalInfo>

Except for the IEC61883 protocol, the 4th field of the *ProtocolInfo* string contains either an asterisk character ("*") or a list of name-value pairs. An asterisk indicates that the 4th field does not contain any meaningful data and should be ignored. A list of name-value pairs indicates that there is some additional information beyond the first three fields. In this case, the 4th field shall contain one or more name-value pairs (separated by a semi-colon ";") with each name-value pair having the following format:

<org-name>_<token-name>=<value>

where

- <org-name> is the ICANN registered domain name of the organization that has defined the semantic of the name-value pair. For example, the UPnP Forum would use an <org-name> of "upnp.org". Case-insensitive comparison is used.

- <token-name> identifies the additional data that is being defined. It consists of one or more alpha-numeric characters (i.e. 'a'-'z', 'A'-'Z', '0'-'9', '_') and shall be unique within the context of the specified <org-name>. Case-insensitive comparison is used.

<value> is the value of the additional data. In addition to the escaping rule defined for attributes in [36] and [37], the following rules also apply:

- All semi-colons (";") within <value> shall be escaped with a backslash ("\"). This is necessary since a semi-colon is used to separate multiple name-value pair occurrences. For example, in order to represent a value of "yours;mine;ours", <value> shall be set to "yours\;mine\;ours".
- All original backslash ("\") characters within <value> shall be escaped with a (second) backslash ("\"). Obviously, backslash characters that have been added as an escape character are themselves not double escaped. For example, in order to represent a value of "yours\mine\ours", <value> shall be set to "yours\\mine\\ours".

Multiple name-value pairs are separated by a semi-colon (";") and have the layout below. When multiple name-value pairs are specified, the order of occurrence of the name-value pair is not relevant. Additionally, the same value of <org-name> may occur multiple times and the same value of <token-name> may occur multiple times. However, each <org-name>_<token-name> combination shall appear at most once within the list of name-value pairs contained by the 4th field. The following example shows three name value pairs: two of which are defined by the UPnP Forum and one of which is defined by a fictitious organization called "VendorA":

upnp.org_resolution=1080i;VendorA.com_resolution=super_high_quality;upnp.org_sample_rate=30FPS

As described before, the <additionalInfo> field may be used for any purpose and contains name-value pairs which the control point may or may not understand. Since the semantics of the unknown name-value pairs are unknown, it should ignore unknown name-value pairs and only known name-value pairs may be used during comparison.

C.2.2 IEC61883 Exception

When the IEC61883 protocol was first introduced into the specification, the structure of the 4th field was not yet defined. Therefore, the additional data that was needed for this protocol was simply placed directly in the 4th field without any higher-level constructs. In order to maintain compatibility with existing implementations of this specification, the definition of the 4th field for the IEC61883 protocol can not be changed in order to comply with the 4th field layout defined above. However, to eventually deprecate this non-conformant protocol designation, a new protocol designation has been defined for IEC61883 which does conform to the above layout. It has been named IEC61883_EX1 with the "_EX1" suffix indicating "Extension #1". All future implementations that support the IEC61883 protocol shall use the new designator (IEC61883_EX1) as well as the original designator (IEC61883).

C.2.3 Formal EBNF for the 4th field

The formal EBNF for the 4th field is as follows:

```
4th-field          ::=  '*'|name-value-pair-list|IEC61883-exception
name-value-pair-list   ::= name-value-pair (';' name-value-pair)*
name-value-pair    ::=  org-name '_' token-name '=' value
org-name           ::=  (* ICANN registered domain name including the top-level
                        domain suffix (e.g. ".com", ".org", ".netv, etc.) *)
token-name         ::=  ('a'-'z' | 'A'-'Z' | '0'-'9' | '_')+
value              ::=  ( unicode-char-except-backslash-semicolon |
                          escaped-backslash |
                          escaped-semicolon
                        )*
unicode-char-except-backslash-semicolon ::= (* any Unicode-4 character except
                        a
                        '\' or '; ' character *)
escaped-backslash  ::=  '\\'

escaped-semicolon  ::=  '\;'
```

```
IEC61883-exception ::=  GUID-value ';' PCR-index

GUID-value         ::=  (* hex encoding of the device's IEC61883 node_vendor_id
                        and chip_id (total of 64-bits) *)

PCR-index          ::=  (* zero-based integer index identifying the plug within
                        the device *)
```

C.2.4 *ProtocolInfo* Conventions for Protected Content

C.2.4.1 3rd Field - MIME Type Format

MIME types for protected content resources appear in the third field, <contentFormat>, of the *ProtocolInfo* string. Since protected files may be described by both a content protection MIME type as well as a MIME type associated with the underlying media, the following convention for extended MIME types will be used for MIME types that describe protected resources:

<content_protection_MIME_type>;CONTENTFORMAT=<underlying_media_MIME_type>

Example 1. The following example describes a MPEG2-TS resource that is protected with DTCP-IP when streaming. The extended MIME type is:

application/x-dtcp1;CONTENTFORMAT=video/MP2T

The full resulting *ProtocolInfo* string additionally indicates that the stream is http-get:

ProtocolInfo = "http-get:*:application/x-dtcp1;CONTENTFORMAT=video/MP2T:*"

Finally, the form of the <res> element content is (per DTCP Volume 1 Supplement E Revision 1.0):

```
<res protocolInfo="http-get:*:application/x-dtcp1;CONTENTFORMAT=video/MP2T:*"
      allowedUse="PLAY,5"
      validityStart="2004-05-30T14:30:00"
      validityEnd="2004-06-04T14:30:00">
   http://10.0.0.1:88/MyCollection/movie7829.mp2t?
   CONTENTPROTECTIONTYPE=DTCP1&amp;DTCP1HOST=1.2.3.4&amp;DTCP1PORT=97
</res>
```

Example 2. The following example extended MIME type describes a MPEG2-PS resource that is delivered as an OMA DCF file:

"application/vnd.oma.drm.dcf;CONTENTFORMAT=video/MP2P"

The resulting *ProtocolInfo* string containing the extended MIME type additionally indicates that the file is transferred with http-get:

ProtocolInfo = "http-get:*:application/vnd.oma.drm.dcf;CONTENTFORMAT=video/MP2P:*"

Finally, the form of the <res> element content is as follows:

```
<res protocolInfo=
   "http-get:*:application/vnd.oma.drm.dcf;CONTENTFORMAT=video/MP2P:*"
   allowedUse="PLAY,5"
   validityStart="2004-05-30T14:30:00"
   validityEnd="2004-06-04T14:30:00">
      http://10.0.0.1:88/MyCollection/movie8126.dcf
</res>
```

Example 3. The following example describes a MPEG2-PS resource that is delivered as an OMA DCF file and can be exported to a DTCP content protection system:

The rights object of the content includes the permissions for enabling the translation into the new DRM system and will be represented as two separate <res> elements representing the same content:

```
<res protocolInfo=
   "http-get:*:application/vnd.oma.drm.dcf;CONTENTFORMAT=video/MP2P:*"
   allowedUse="PLAY,5"
   validityStart="2004-05-30T14:30:00"
   validityEnd="2004-06-04T14:30:00">
      http://10.0.0.1:88/MyCollection/movie8126.dcf
</res>

<res protocolInfo=
   "http-get:*:application/x-dtcp1;CONTENTFORMAT=video/MP2T:*"
   allowedUse="PLAY,5"
   validityStart="2004-05-30T14:30:00"
   validityEnd="2004-06-04T14:30:00">
      http://10.0.0.1:88/MyCollection/movie9736.mp2t?
      CONTENTPROTECTIONTYPE=DTCP1&amp;DTCP1HOST=1.2.3.4&amp;DTCP1PORT=97
</res>
```

C.2.4.2 4th Field - Convention for Protected Content

The UPnP AV WC additionally defines the following convention for placing information in the fourth field of the *ProtocolInfo* string. The fourth field is used to convey additional information in cases when MIME type is not sufficient for the purpose of capability matching. In these cases the fourth field shall contain additional DRM information for the purpose of more precise compatibility checking between the media sink and the content properties. Two cases are possible. In the first case, it is only required to identify the vendor or the standards group that defines the DRM scheme. In this case, the DRM organization or vendor is specified as the value of a upnp-domain variable:

```
upnp.org_DRMInfo=<ICANN-DRM-ORG-NAME>
```

In the second case, the DRM scheme also requires one or more parameters associated with the DRM scheme to be enumerated. In this case, the following convention is used:

```
upnp.org_DRMInfo=<ICANN-DRM-ORG-NAME>;
<ICANN-DRM-ORG-NAME>_<parameter1>=<parameter1 value>;
<ICANN-DRM-ORG-NAME>_<parameter2>=<parameter2 value>;
…
<ICANN-DRM-ORG-NAME>_<parameterN>=<parameterN value>
```

Example 4: A *ProtocolInfo* string utilizes the fourth field to indicate that the MPEG-4 content is protected by a (fictitious) DRM scheme associated with ICANN name “XYZ.ORG”:

```
ProtocolInfo = “http-get:*:video/mp4:upnp.org_DRMInfo=XYZ.ORG”
```

Example 5: A *ProtocolInfo* string for an OMA dcf file also indicates the OMA version in the fourth field:

```
ProtocolInfo = “http-get:*:application/vnd.oma.drm.dcf;
               CONTENTFORMAT=video/MP2P:upnp.org_DRMInfo=OMA.ORG;OMA.ORG_VERSION=2”
```

For maximal compatibility checking, both third and fourth fields (when present) of the *ProtocolInfo* string should be matched. In general, older control points may not be capable of checking the fourth field of the *ProtocolInfo* string. For that reason, it is recommended that content listings in the CDS for DRM content should use the MIME conventions described above as much as possible for inserting DRM information on the given content item into the third field of the *ProtocolInfo* string.

C.3 Typical Control Point Operations

C.3.1 Introduction

Annex C.3 briefly outlines some typical control point operations on a ConnectionManager service.

C.3.2 Establishing a New Connection

The process for establishing a streaming connection involves:

a) Find ConnectionManager services via SSDP

b) Determine compatibility between the source (sending) and the sink (receiving) device (see subclause 5.4.2).

c) Invoke *PrepareForConnection()* on the source and/or sink devices, if the action is implemented by the device (see subclause 5.4.3).

d) Transfer content from source to sink device (see note below).

e) When the connection is no longer needed, invoke *ConnectionComplete()* on the source and/or sink devices, if the action is implemented by the device (see subclause 5.4.4).

Refer to the UPnP AV Architecture document [47] for additional details.

Once a connection has been prepared, it can be used to transfer several pieces of content before calling *ConnectionComplete()* as long as each content item is compatible with the *RemoteProtocolInfo* argument that was passed into *PrepareForConnection()*; that is: each content item has the same media format as specified in *RemoteProtocolInfo*.

C.3.3 Dealing with Ongoing Connections

A number of interesting scenarios require a control point to find information about all currently ongoing connections in the network, including those that it did not establish itself. This is supported by the ConnectionManager as follows. Each connection explicitly established by any control point in the network is identified by a connection identifier on both the source (sending) device and the sink (receiving) device. State variable *CurrentConnectionIDs* holds a Comma-Separated Value list of these identifiers. Given an identifier, a control point can call *GetConnectionInfo()* to obtain:

a) The *ProtocolInfo* of the connection. This includes the streaming protocol and the content format.

b) The 'other end' of the connection, expressed as a **UDN**/**serviceId** pair. Using the **UDN**, a control point can use SSDP to find the device description of the other UPnP device involved in the connection. This way, a control point can find out, for example, that turning off a particular source device is going to affect one or more sink devices.

c) The connection status.

d) The *AVTransportID* of the connection, which indicates the AVTransport service instance controlling the playback and recording through the connection. This service can be used for many purposes, for example to:

   1) subscribe to events in order to monitor the transport state

   6) actually change the transport state, for example, stopping or pausing an existing stream

   7) obtain a URI reference to the content resource currently flowing through the connection

   8) obtain any meta data embedded in the content resource flowing through the connection.

   See the AVTransport service description [5] for more details.

e) The *RcsID* of the connection, which indicates the RenderingControl service instance controlling the rendering properties of the content. This can be used, for example, to implement a 'mute all streams' function in a control point.

C.4 Relation to Devices without ConnectionManagers

In some cases, it is desirable to establish a stream connection between devices where one device implements a UPnP ConnectionManager service, and the other device doesn't implement this service or isn't even a UPnP device. In such cases, a control point can only call *PrepareForConnection()* and *ConnectionComplete()* actions on the first device. The *PeerConnectionManager* input argument to *PrepareForConnection()* is defined as the **UDN** of the connecting UPnP device followed by a slash ('/') and the **serviceId** of the connecting

device's ConnectionManager service. In case the connecting UPnP device has no ConnectionManager service, the **serviceId** part of the argument is left blank. In case the connecting device is not a UPnP device (for example, an Internet streaming server), the whole PeerConnectionManager argument is left blank.

C.5 *PrepareForConnection()* and *ConnectionComplete()*

C.5.1 *PrepareForConnection()*

The purpose of *PrepareForConnection()* is to allow a device to perform a set of tasks prior to transferring the content. The specific tasks performed by a device are implementation dependent, but may include the following:

a) Determine whether or not the device is able to stream content using the current environment (for example, device status, network conditions, etc.)

a) Allocating some resources that are needed to establish the out-of-band connection between the source/sink devices for example, in an IEEE-1394/IEC-61883 environment, this may include allocating an IEEE-1394 isochronous channel.

b) Allocating a unique *ConnectionID* that identifies those resources which were allocated for a specific connection.

c) Allocating a new virtual instance of the AVTransport and/or RenderingControl service and binding it to the connection so that the flow of the content and the rendering characteristics of the content can be controlled.

If a control point wants to interoperate with all UPnP AV devices, prior to initiating the transfer of content, for example, invoking *AVTransport::SetAVTransportURI()*, the control point needs to invoke *PrepareForConnection()*, if the action is implemented by the device. Otherwise, the device may not operate correctly because it is not yet properly configured. Additionally, the control point will not know whether or not the current environment is able to support the upcoming streaming request.

C.5.2 *ConnectionComplete()*

The purpose of *ConnectionComplete()* is to allow a device to terminate a specific connection and/or to perform any cleanup tasks that are needed for the connection as a result of a previous invocation of *PrepareForConnection()*. As with *PrepareForConnection()*, the set of tasks performed by a device when *ConnectionComplete()* is invoked is implementation-dependent, but may include the following:

a) Releasing the resources that were allocated to establish the out-of-band connection between the source and sink devices (for example, in a IEEE-1394/IEC-61883 environment, this may include releasing the IEC-61883 isochronous channel that was allocated when *PrepareForConnection()* was invoked earlier on the same device).

b) Releasing the unique *ConnectionID* that identifies those resources that were allocated by a previous invocation of *PrepareForConnection()*.

c) Releasing the virtual instance of the AVTransport and/or RenderingControl service, if any, that were allocated during a previous invocation of *PrepareForConnection()* in order to control the content flowing over the associated connection.

Since control points may turn off after a connection is established, control points may not always invoke the *ConnectionComplete()* action. Therefore, the device needs to automatically perform any cleanup tasks for the connection so that those resources that were allocated during *PrepareForConnection()* are not *leaked*.

C.5.3 General Usage Model

As mentioned earlier, each device performs an arbitrary set of implementation-dependent tasks during *PrepareForConnection()* and *ConnectionComplete()*. Some of these tasks may be crucial to the proper operation of the device while other tasks may be secondary to the device's core functionality. However, since each implementation of *PrepareForConnection()* and *ConnectionComplete()* are specific to each device, it is very difficult (if not impossible) for a control point to determine whether or not it is safe to by-pass

PrepareForConnection()/ConnectionComplete() for a given device. Therefore, the safest and simplest way for a control point to interoperate with all UPnP AV devices is to always invoke PrepareForConnection() and ConnectionComplete() if they are implemented by the device. Otherwise, those devices may not function properly as described above.

C.5.4 Relationship to AVTransport and RenderingControl Services

As described in the "Theory of Operation" annexes of the AVTransport [5] and RenderingControl [21] service specifications, some device are designed to support multiple virtual instances of the AVTransport and/or RenderingControl service. With these types of devices, the allocation and binding of these virtual instances occur during PrepareForConnection().

As described in the AVTransport specification, the responsibility for providing the AVTransport service for a given connection varies between the source and sink devices depending on the type of connection (that is: the type of transfer protocol that is being used). When a *push* protocol is being used (for example, IEEE-1394), the source device is responsible for providing the AVTransport service and when a *pull* protocol is being used (for example, HTTP GET), the sink device is responsible for providing the AVTransport service. When a source device wants to support multiple instances of a *push* protocol or a sink device wants to support multiple instances of a *pull* protocol, the device's PrepareForConnection() is responsible for allocating a new virtual instance of the AVTransport service for each new instance of that connection-type. Additionally, PrepareForConnection() shall perform the necessary binding operations that link the allocated AVTransport instance with the connection.

For example, in a IEEE-1394/IEC-61883 (*push*) environment, if a source device wants to support multiple 1394/61883 streams, then its PrepareForConnection() shall allocate a unique virtual instance of the AVTransport service and bind it to the newly allocated IEEE-1394/IEC-61883 connection. Similarly, in an HTTP GET *pull* environment, if a sink device wants to support multiple simultaneous connections, its PrepareForConnection() implementation shall allocate a new virtual instance of the AVTransport service and bind it to the newly allocated connection.

Note: This implies that the sink device's PrepareForConnection() shall perform some type of pre-allocation of the TCP/IP socket so that it can be distinguished from the other connections of that type.

With regards to the RenderingControl service, the sink device is always responsible for providing it regardless of the underlying protocol. If a sink device is designed to support multiple simultaneous connections then its implementation of PrepareForConnection() shall be designed to allocate a new virtual instance of the RenderingControl service for each connection that is created. Additionally, it shall bind each instance to the newly created connection so that a control point can control the rendering characteristics of the content that is being transferred over that connection.

Note: This implies that the sink device's PrepareForConnection() shall perform some type of pre-allocation of the TCP/IP socket so that it can be distinguished from the other connections of that type.

When a device's PrepareForConnection() is designed to allocate and bind virtual instances of the AVTransport and/or RenderingControl services, the device's ConnectionComplete() shall be designed to un-bind and release these virtual instances. For example, if a source device allocates an AVTransport service and binds it to a IEEE-1394 channel, the device's ConnectionComplete() action shall undo the binding operation, as appropriate, and it shall release the IEEE-1394 channel.

C.5.5 ***ConnectionIDs***

A ConnectionID is a device-specific identifier that is used to uniquely identify a connection which has been prepared on a device via PrepareForConnection(). When PrepareForConnection() is invoked, a ConnectionID is allocated by the device and assigned to the newly configured connection. Since ConnectionIDs are allocated by individual devices, a given ConnectionID is valid only within the context of that device. Therefore, a ConnectionID assigned by one device cannot be used when interacting with another device. When the two end-point devices of a

given connection are setup via *PrepareForConnection()*, the *ConnectionID*s returned by the two devices are completely independent from each other and are almost certainly going to have different values even though they happen to refer to the same connection.

On a given device, the algorithm used to allocate *ConnectionID*s is vendor-specific. Hence, the numerical value of a *ConnectionID* is completely meaningless except to the device itself. Once a *ConnectionID* has been allocated, it is generally valid until the associated connection is torn down. Typically, this happens in response to an invocation of *ConnectionComplete()* or as a result of the device's allowed auto-cleanup mechanism.

Devices should not return the same *ConnectionID* value on subsequent invocations of *PrepareForConnection()*. After a connection has been torn down, its associated *ConnectionID*, which is no longer valid, can be reassigned by the device to another connection. However, in order to minimize the potential of a *stale* *ConnectionID* being misinterpreted as a *valid* *ConnectionID*, it is recommended that each device not reassign a *ConnectionID* value until all other valid values have been used.

Once a *ConnectionID* has been allocated, any control point may use the *ConnectionID* to uniquely identify the associated connection even when invoking *ConnectionComplete()*. However, in order to provide predictable device behavior, it is recommended that each control point use only those connections that it has prepared. Notable exceptions to this recommendation include those control points that are able to coordinate with one another, via some non-UPnP mechanism, or those control points that are explicitly designed to perform connection clean up tasks, for example, a network management tool.

C.5.6 AVTransportIDs and RcsIDs

When a connection is prepared via *PrepareForConnection()*, the device may choose to return an *AVTransportID* and/or an *RcsID*. These IDs are used to identify the (unique and independent) virtual instance of the AVTransport service and/or RenderingControl service that has been associated with the newly prepared connection. As with the connection's *ConnectionID*, the value of the *AVTransportID* and/or *RcsID* have no meaning outside of the context of the allocating device. Similarly, *AVTransportID*s and *RcsID*s are valid until their associated connection is torn down, generally in response to an invocation of *ConnectionComplete()* or as a result of the device's allowed auto-cleanup mechanism.

Once allocated, *AVTransportID*s and *RcsID*s are used in conjunction with the AVTransport service and RenderingControl service, respectively, to invoke various control actions on the stream that is being carried over the associated connection. As with *ConnectionID*s, control points should use only the *AVTransportID*s and *RcsID*s that are associated with the connections that the control point has prepared via *PrepareForConnection()*. The AVTransport and RenderingControl services allow any control point to use any valid *AVTransportID*s and/or *RcsID*s. However, when multiple control points use the same *AVTransportID* and/or *RcsID*, these control points should coordinate their activities with one another. Otherwise, the devices may behave unexpectedly thus causing a poor end-user experience.

C.6 Determining if ContentDirectory items are playable

The *GetRendererItemInfo()* action allows a control point to request the rendering device inspect item metadata provided and determine if the rendering device expects to be able to successfully play the item described. The MediaRenderer control point issuing this action submits one or more items obtained from a MediaServer device using the ContentDirectory service's *Browse()* or *Search()* actions. Each item will normally contain one or more *res* properties that specify "content-binaries" that the MediaRenderer device may play.

The control point may use the *ItemInfoFilter* argument to control the amount of information returned for the item(s) submitted.

The following informative examples are provided:

Example 1: The *GetRendererItemInfo()* action is issued with an empty filter argument indicating only basic information should be returned.

```
Request:
GetRendererItemInfo("",
"<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">
      <dc:title>Try a little tenderness</dc:title>
      <upnp:class>object.item.audioItem.musicTrack</upnp:class>
      <upnp:longDescription>
         This song is considered to be the finest R&amp;B tune ever
      </upnp:longDescription>
      <dc:creator>Otis Redding</dc:creator>
      <res protocolInfo="http-get:*:audio/L16;rate=44100;channels=2:*"
            bitrate="6553"
            nrAudioChannels="2"
            duration="03:12"
            size="1258291">
         http://10.0.0.1/audio/O-l16-211.pcm
      </res>
      <res  protocolInfo="http-get:*:audio/mpeg:*"
            bitrate="6553"
            nrAudioChannels="2"
            duration="03:12"
            size="8291">
         http://10.0.0.1/audio/O-MP3-211.mp3
      </res>
      <res protocolInfo="http-get:*:audio/wma:*"
            bitrate="6553"
            nrAudioChannels="2"
            duration="03:12"
            size="58291">
         http://10.0.0.1/audio/O-WMA-211.wma
      </res>
   </item>
</DIDL-Lite>" )
```

The ConnectionManager service responds with a *RendererInfo XML document* that indicates the rendering device capability to play each *DIDL-Lite* item described ItemMetadataList argument:

```
Response:
GetRendererItemInfo("
<?xml version="1.0" encoding="UTF-8"?>
<rendererInfo xmlns="urn:schemas-upnp-org:av:rii"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="
   urn:schemas-upnp-org:av:rii
    http://www.upnp.org/schemas/av/rii.xsd">
        <itemInfo itemID="18">
            <resPlaybackInfo resIndex="0" canPlay="1" />
            <resPlaybackInfo resIndex="1" canPlay="1" />
            <resPlaybackInfo resIndex="2" canPlay="0" />
       </itemInfo>
    </rendererInfo>" );
```

Example 2: The *GetRendererItemInfo()* action is issued with a filter indicating that DRM related information should be returned:

```
Request:
GetRendererItemInfo(
```

```
"itemInfo::resPlaybackInfo::drmInfo#",
"<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">
      <dc:title>Try a little tenderness</dc:title>
      <upnp:class>object.item.audioItem.musicTrack</upnp:class>
      <upnp:longDescription>
         This song is considered to be the finest R&amp;B tune ever
      </upnp:longDescription>
      <dc:creator>Otis Redding</dc:creator>
      <res protocolInfo="http-get:*:audio/L16;rate=44100;channels=2:*"
            bitrate="6553"
            nrAudioChannels="2"
            duration="03:12"
            size="1258291">
         http://10.0.0.1/audio/O-l16-211.pcm
      </res>
   </item>
   <item id="19" parentID="13" restricted="0">
      <dc:title>Try a little tenderness</dc:title>
      <upnp:class>object.item.audioItem.musicTrack</upnp:class>
      <upnp:longDescription>
         This song is considered to be the finest R&amp;B tune ever
      </upnp:longDescription>
      <dc:creator>Otis Redding</dc:creator>
      <res protocolInfo="http-get:*:audio/wav:*"
            bitrate="6553"
            nrAudioChannels="2"
            duration="03:12"
            size="1258291">
         http://10.0.0.1/audio/O-l16-212.wav
      </res>
   </item>
   <item id="20" parentID="13" restricted="0">
      <dc:title>Try a little tenderness</dc:title>
      <upnp:class>object.item.audioItem.musicTrack</upnp:class>
      <upnp:longDescription>
         This song is considered to be the finest R&amp;B tune ever
      </upnp:longDescription>
      <dc:creator>Otis Redding</dc:creator>
      <res protocolInfo="http-get:*:audio/l16;rate=44100;channels=2:*"
            bitrate="6553"
            nrAudioChannels="2"
            duration="03:12"
            size="1258291">
         http://10.0.0.1/audio/O-l16-213.pcm
      </res>
   </item>
</DIDL-Lite>" )
```

Response:

```
GetRendererItemInfo("
<?xml version="1.0" encoding="UTF-8"?>
<rendererInfo xmlns="urn:schemas-upnp-org:av:rii"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="
   urn:schemas-upnp-org:av:rii
    http://www.upnp.org/schemas/av/rii.xsd">
    <itemInfo itemID="18">
```

```
        <resPlaybackInfo resIndex="0" canPlay="1">
            <drmInfo drmProtected="1" drmStatus="OK">
                <drmSystem>
                    <friendlyName>Not So Free Music</friendlyName>
          <ICANNName>OpenMobileAlliance.ORG</ICANNName>
                    <systemName>DRM_REL_DD</systemName>
                    <systemVersion>2.1</systemVersion>
                </drmSystem>
                <licenseIdentifier>
                    uuid:550e8400-e29b-41d4-a716-446655440000
                </licenseIdentifier>
                <licenseRights type="play">
                   <licenseUsageTimeRemaining>
    P5D20:00:00
</licenseUsageTimeRemaining>
                   <licenseSubscriptionTimeRemaining>
                          P1D20:00:00
                   </licenseSubscriptionTimeRemaining>
                   <licenseUsageCountRemaining>4</licenseUsageCountRemaining>
            </licenseRights>
              </drmInfo>
        </resPlaybackInfo>
</itemInfo>
    <itemInfo itemID="19">
        <resPlaybackInfo resIndex="0" canPlay="0">
            <drmInfo drmProtected="1" drmStatus="LICENSE_EXPIRED">
                <drmSystem>
                    <friendlyName>Not So Free Music</friendlyName>
          <ICANNName>OpenMobileAlliance.ORG</ICANNName>
                    <systemName>DRM_REL_DD</systemName>
                    <systemVersion>2.1</systemVersion>
                </drmSystem>
</drmInfo>
        </resPlaybackInfo>
    </itemInfo>
    <itemInfo itemID="20">
        <resPlaybackInfo resIndex="0" canPlay="0" >
            <drmInfo drmProtected="1" drmStatus="LICENSE_DENIED">
                <drmSystem>
                    <friendlyName>Not So Free Music</friendlyName>
          <ICANNName>OpenMobileAlliance.ORG</ICANNName>
                    <systemName>DRM_REL_DD</systemName>
                    <systemVersion>2.1</systemVersion>
                </drmSystem>
</drmInfo>
        </resPlaybackInfo>
    </itemInfo>
</rendererInfo>" )
```

Example 3: The *GetRendererItemInfo()* action is issued with a filter indicating that only playback related information should be returned:

```
Request:
GetRendererItemInfo(
"itemInfo::resPlaybackInfo::playbackInfo#",
"<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">
      <dc:title>Try a little tenderness</dc:title>
      <upnp:class>object.item.videoItem</upnp:class>
```

```
      <res protocolInfo="http-get:*:video/mpeg:*">
         http://10.0.0.1/video/test_video.mpg
      </res>
   </item>
    <item id="18" parentID="13" restricted="0">
      <dc:title>Try a little tenderness</dc:title>
      <upnp:class>object.item.imageItem</upnp:class>
      <res protocolInfo="http-get:*:image/jpeg:*">
         http://10.0.0.1/image/test_image.jpg
      </res>
   </item>
  </DIDL-Lite>" )

Response:
GetRendererItemInfo("
<?xml version="1.0" encoding="UTF-8"?>
<rendererInfo xmlns="urn:schemas-upnp-org:av:rii"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="
   urn:schemas-upnp-org:av:rii
    http://www.upnp.org/schemas/av/rii.xsd">
    <itemInfo itemID="18">
        <resPlaybackInfo resIndex="0" canPlay="1">
        <playbackInfo playbackCompatibility="OK">
                <videoStreamInfo outputResolution="480i" />
                <audioStreamInfo outputChannels="7.1" />
        </playbackInfo>
         </resPlaybackInfo>
    </itemInfo>
    <itemInfo itemID="19">
        <resPlaybackInfo resIndex="0" canPlay="1">
        <playbackInfo playbackCompatibility="OK">
            <imageStreamInfo outputResolution="640x480p" />
        </playbackInfo>
         </resPlaybackInfo>
    </itemInfo>
</rendererInfo>" )
```

Example 4: The *GetRendererItemInfo()* action is issued on an item with a resource containing multiple components with a filter indicating that only transform related information will be returned:

```
Request:
GetRendererItemInfo(
"itemInfo::resPlaybackInfo::transformInfo#",
"<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="100" parentID="200" restricted="0">
   <dc:title>KBS News</dc:title>
   <upnp:class>object.item.videoItem</upnp:class>
   <res id="100-res-1" protocolInfo="http-get:*:video/mpeg:*"
resolution="1920x1080">
      http://10.0.0.1/content/content?id=100-res
   </res>
  <upnp:resExt id="100-res-1">
     <upnp:isSyncAnchor>1</upnp:isSyncAnchor>
     <upnp:componentInfo>
       <upnp:componentGroup groupID="0">
         <upnp:component componentID="comp_0">
          <upnp:componentClass>Video</upnp:componentClass>
```

```
            <upnp:purpose>Default</upnp:purpose>
            <upnp:contentType MIMEType=”video/MPV” extendedType=”*”/>
          </upnp:component>
       </upnp:componentGroup>
       <upnp:componentGroup groupID="1"
         <upnp:component componentID="comp_1">
            <upnp:componentClass>Audio</upnp:componentClass>
           <upnp:purpose>Default</upnp:purpose>
           <upnp:language>en-US</upnp:language>
           <upnp:contentType MIMEType=”audio/ac3” extendedType=”*”/>
         </upnp:component>
         <upnp:component componentID="comp_2">
             <upnp:componentClass>Audio</upnp:componentClass>
           <upnp:purpose>Alternative</upnp:purpose>
           <upnp:language>fr</upnp:language>
           <upnp:contentType MIMEType=”audio/MPA” extendedType=”*”/>
         </upnp:component>
      </upnp:componentGroup>
      <upnp:componentgroup groupID="2">
         <upnp:component componentID="comp_5">
             <upnp:componentClass>Caption</upnp:componentClass>
          <upnp:purpose>Alternative</upnp:purpose>
           <upnp:language>nl</upnp:language>
           <upnp:contentType MIMEType=”text/srt” extendedType=”*”/>
         </upnp:component>
         <upnp:component componentID="comp_6">
             <upnp:componentClass>Caption</upnp:componentClass>
          <upnp:purpose>Alternative</upnp:purpose>
           <upnp:language>de</upnp:language>
           <upnp:contentType MIMEType=”text/sub” extendedType=”*”/>
         </upnp:component>
       </upnp:componentGroup>
     </upnp:componentInfo>
   </upnp:resExt>
</item>
</DIDL-Lite>")

Response:
GetRendererItemInfo("
<?xml version="1.0" encoding="UTF-8"?>
<rendererInfo xmlns=" urn:schemas-upnp-org:av:rii"
    xsi:schemaLocation="
   urn:schemas-upnp-org:av:rii
    http://www.upnp.org/schemas/av/rii.xsd">
    <itemInfo itemID="100">
        <resPlaybackInfo resIndex="0" canPlay="1">
      <transformInfo>
         <TransformList
            xmlns="urn:schemas-upnp-org:av:AllowedTransformSettings"
            xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
            xsi:schemaLocation=
               "urn:schemas-upnp-org:av:AllowedTransformSettings
               http://www.upnp.org/schemas/av/avs.xsd">
            <transform name="AudioTrackSelection">
               <allowedValueList>
                  <value>en-US</value>
                  <value>fr</value>                              </allowedValueList>
            </transform>
         </TransformList>
      </transformInfo>
   </resPlaybackInfo>
</itemInfo>
</rendererInfo>")
```

From the response, it can be concluded that the MediaRenderer is able to play back this item, and it is capable of change the audio track language dynamically. However, it is not capable of changing the language for the closed captioning.

C.7 CLOCKSYNC feature

A control point invokes GetFeatureList() action (on a MediaRenderer's or MediaServer's ConnectionManager service) to determine the synchronized playback mechanisms supported by the device. All of the *CLOCKSYNC feature* information is encapsulated within a <Feature> element, which further contains <deviceClockInfo> elements. Each <deviceClockInfo> element describes a clock synchronization mechanism that is available on the device. For a MediaRenderer, the presence of a <deviceClockInfo> element indicates the MediaRenderer's ability to play content using that synchronization mechanism. For a MediaServer, the presence of a <deviceClockInfo> element indicates the MediaServer's ability to provide content using that synchronization mechanism.

Even if a control point is able to match the same <deviceClockInfo> data from the ConnectionManager services of a MediaRenderer and a MediaServer, synchronized playback is not guaranteed for all of the content advertised by the MediaServer's ContentDirectory service. Matching <deviceClockInfo> data acquired from the ConnectionManager service on each endpoint only indicates that synchronized playback might be possible.

Synchronized playback becomes possible when the control point is able to match a variety of things. Specifically, a proper match exists only when these conditions are met:

a) The MediaRenderer's <deviceClockInfo> data matches against a MediaServer's <deviceClockInfo> data. Specifically, the values of <syncProtocolID>, <masterClockID>, and <supportedTimestamps> need to match between the devices. The supportedTimestamps@protocol and supportedTimestamps@format on both endpoints must also match using comparison conventions established for the first and third fields of protocolInfo values.

b) Given a matched <deviceClockInfo> from both MediaRenderer and MediaServer, the MediaRenderer's supportedTimestamps@protocol and supportedTimestamps@format (from the matched <deviceClockInfo>) must also match the first and third fields of the content's res@protocol, respectively.

c) Lastly, the content's <upnp:clockSync> information (acquired from a MediaServer's ContentDirectory service) must reference the same MediaServer's <deviceClockInfo> data. Specifically the upnp:clockSync@deviceClockInfoID must also match against the deviceClockInfo@id and upnp:clockSync@supportedTimestampsID must also match against the supportedTimestamps@id. See Annex D.21 in ContentDirectory service [7] for more information about comparing <deviceClockInfo> against <upnp:clockSync>.

Within the <deviceClockInfo> element, the <syncProtocolID> element indicates a clock synchronization protocol that is available for synchronizing the device's local time-of-day clock. The possible clock synchronization protocols include 802.1AS, NTP (Network Timing Protocol) and SNTP (Simple Network Timing Protocol) protocols. The 802.1AS clock synchronization protocol enables precision synchronization (accuracy better than 1 micro-second), thus enabling usages such as synchronized audio & video playback. For other usages such as party music being piped to multiple rooms, NTP and SNTP protocols most likely can provide sufficient clock synchronization accuracy.

Similarly, the <masterClockID> element (also within the <deviceClockInfo> element) of the *CLOCKSYNC feature* identifies the master clock to which this implementation has synchronized its local time-of-day clock. Depending on the clock synchronization protocol, the <masterClockID> element specifies either the 8-byte binary sequence (<High 24-bits MAC> 0xFF 0xFE <Low 24-bits MAC>) in case of 802.1AS, or the URL of the time server in case of NTP or SNTP.

C.7.1 Examples of *CLOCKSYNC* feature

The FeatureList state variable is obtained via the GetFeatureList() action, and the examples below describe responses from two different devices.

Request to Device0:

```
GetFeatureList()
```

Response from Device0:

```
GetFeatureList("
<Features
 xmlns="urn:schemas-upnp-org:av:cm-featureList"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="
   urn:schemas-upnp-org:av:cm-featureList
    http://www.upnp.org/schemas/av/cm-featureList.xsd">
   <!-- No clock devices defined for this device -->
</Features>"
)
```

Request to Device1:

```
GetFeatureList()
```

Response from Device1:

```
GetFeatureList("
<Features
 xmlns="urn:schemas-upnp-org:av:cm-featureList"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="
   urn:schemas-upnp-org:av:cm-featureList
    http://www.upnp.org/schemas/av/cm-featureList.xsd">
   <Feature name="CLOCKSYNC" version="1">
      <!-- Clock info for the local device -->
      <deviceClockInfo id="Unique ID for Device1" updateID=”1”>
         <syncProtocolID>802.1AS</syncProtocolID>
         <masterClockID>123456FFFE789ABC</masterClockID>
         <accuracy>100</accuracy>
         <!Timestamps supported for RTP with all media formats -->
         <supportedTimestamps
          id="123" protocol="rtsp-rtp-udp" format="*">
            RTP+IEEE-1733
         </supportedTimestamps>
      </deviceClockInfo>
   </Feature>
</Features>"
)
```

In the above example, Device0 does not support any clock synchronization feature, and hence this device is not capable of being synchronized in time with other devices on the network. This generalized example can apply when Device0 is a MediaServer and Device1 is a MediaRenderer, or vice versa.

The *Features* state variable for Device1 indicates that Device1 supports the IEEE-802.1AS (802.1AS) clock synchronization protocol and is synchronized to the clock master whose ID is 123456FFFE789ABC. Device1 guarantees to remain within 100 nanoseconds of the clock master. Additionally, if Device1 is a MediaRenderer, then it is able to support synchronized playback of *rtsp-rtp-udp* content stream of any format via the IEEE-1733 (RTP) timestamping mechanism. If Device1 is a MediaServer, then it might be able to support synchronized playback of some of its *rtsp-rtp-udp* content streams via the IEEE-1733 (RTP) timestamping mechanism.

The ConnectionManager service’s `<Features>` element should not list both the Identity and RTP+IEEE-1733 Time Stamp Mechanisms using the same `<deviceClockInfo>` element. This avoids the possibility of different MediaRenderers attempting synchronized playback using incompatible Time Stamp Mechanisms.

The next example shows a `<Features>` element of a device that supports both the RTP+IEEE-1733 and Identity Time Stamp mechanisms:

Request:

```
GetFeatureList()
```

Response (with RTP+IEEE-1733 and Identity):

```
GetFeatureList("
<Features
 xmlns="urn:schemas-upnp-org:av:cm-featureList"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="
   urn:schemas-upnp-org:av:cm-featureList
    http://www.upnp.org/schemas/av/cm-featureList.xsd">
   <Feature name="CLOCKSYNC" version="1">
      <deviceClockInfo id="ClockInfo-1" updateID=”1”>
         <syncProtocolID>802.1AS</syncProtocolID>
         <masterClockID>123456FFFE789ABC</masterClockID>
         <accuracy>1</accuracy>
         <supportedTimestamps
          id="Timestamp-1" protocol="rtsp-rtp-udp" format=”*">
            RTP+IEEE-1733
         </supportedTimestamps>
      </deviceClockInfo>
      <deviceClockInfo id="ClockInfo-2" updateID=”1”>
         <syncProtocolID>802.1AS</syncProtocolID>
         <masterClockID>123456FFFE789ABC</masterClockID>
         <accuracy>1</accuracy>
         <supportedTimestamps
          id="Timestamp-1" protocol="rtsp-rtp-udp" format=”*">
            Identity
         </supportedTimestamps>
         <supportedTimestamps
          id="Timestamp-2" protocol="http-get" format="*">
            Identity
         </supportedTimestamps>
      </deviceClockInfo>
   </Feature>
</Features>"
)
```

Annex D
(informative)

Bibliography

The following documents, in whole or in part, may be useful for understanding this document but they are not essential for its application. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

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

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

[50] – *DeviceProtection:1*, UPnP Forum, February 24, 2011.
Available at: http://www.upnp.org/specs/gw/UPnP-gw-DeviceProtection-v1-Service-20110224.pdf.
Latest version available at: http://www.upnp.org/specs/gw/UPnP-gw-DeviceProtection-v1-Service.pdf.

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

[52] – *MediaRenderer:3*, UPnP Forum, March 31, 2013.
Available at: http://www.upnp.org/specs/av/UPnP-av-MediaRenderer-v3-Device-20130331.pdf.
Latest version available at: http://www.upnp.org/specs/av/UPnP-AV-MediaRenderer-v3-Device.pdf.

[53] – *MediaServer:4*, UPnP Forum, March 31, 2013.
Available at: http://www.upnp.org/specs/av/UPnP-av-MediaServer-v4-Device-20130331.pdf.
Latest version available at: http://www.upnp.org/specs/av/UPnP-AV-MediaServer-v4-Device.pdf.

[54] – *IETF RFC 1321, The MD5 Message-Digest Algorithm*, R. Rivest, April 1992.
Available at: http://tools.ietf.org/html/rfc1321.

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

[56] – *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.

[57] – *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.

[58] – *IETF RFC 3986, Uniform Resource Identifiers (URI): Generic Syntax*, January 2005.
Available at: http://www.ietf.org/rfc/rfc3986.txt.

[59] – *IETF RFC 3174, US Secure Hash Algorithm 1 (SHA1)*, D. Eastlake et al, September 2001.
Available at: http://tools.ietf.org/html/rfc3174.

[60] – *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.

[61] – *IETF RFC 4078, The TV-Anytime Content Reference Identifier (CRID)*, N. Earnshaw et al, May 2005.
Available at: http://www.ietf.org/rfc/rfc4078.txt.

[62] – *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.

[63] – *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.

[64] – *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.

[65] – *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.

[66] – *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.

[67] – *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.

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