



# UPnP Device Architecture 2.0

Document Revision Date: February 20, 2015

THIS APPROVED SPECIFICATION WAS COMPLETED PRIOR TO THE COMBINATION OF UPNP INTO THE OPEN CONNECTIVITY FOUNDATION. ALL LICENSES, INTELLECTUAL PROPERTY RIGHTS, AND OTHER RIGHTS, RESPONSIBILITIES, OBLIGATIONS, STANDARDS, AND PROTOCOLS ASSOCIATED WITH THIS APPROVED SPECIFICATION ARE SUBJECT TO THE UPNP BYLAWS AND FORUM MEMBERSHIP AGREEMENT.

## Legal Disclaimer

NOTHING CONTAINED IN THIS DOCUMENT SHALL BE DEEMED AS GRANTING YOU ANY KIND OF LICENSE IN ITS CONTENT, EITHER EXPRESSLY OR IMPLIEDLY, OR TO ANY INTELLECTUAL PROPERTY OWNED OR CONTROLLED BY ANY OF THE AUTHORS OR DEVELOPERS OF THIS DOCUMENT. THE INFORMATION CONTAINED HEREIN IS PROVIDED ON AN "AS IS" BASIS, AND TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, THE AUTHORS AND DEVELOPERS OF THIS SPECIFICATION HEREBY DISCLAIM ALL OTHER WARRANTIES AND CONDITIONS, EITHER EXPRESS OR IMPLIED, STATUTORY OR AT COMMON LAW, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. OPEN CONNECTIVITY FOUNDATION, INC. FURTHER DISCLAIMS ANY AND ALL WARRANTIES OF NON-INFRINGEMENT, ACCURACY OR LACK OF VIRUSES.

The OCF Logo, UPnP Word Mark and UPnP Logo are trademarks of Open Connectivity Foundation, Inc. in the United States or other countries. \*Other names and brands may be claimed as the property of others.

Copyright © 2016 Open Connectivity Foundation, Inc. All rights reserved.

Copying or other form of reproduction and/or distribution of these works are strictly prohibited.

<b>Authors*</b>	<b>Company</b>
Andrew Donoho	IBM
Bryan Roe	Intel
Maarten Bodlaender	Philips
John Gildred	Pioneer
Alan Messer	Samsung
YoonSoo Kim	Samsung
Bruce Fairman	Sony
Jonathan Tourzan	Sony

**\*Note:** The UPnP Forum in no way guarantees the accuracy or completeness of this contributor list and in no way implies any rights for or support from those members listed.

<b>Contributors*</b>	<b>Company</b>
Alan Presser	AllegroSoft
Devon Kemp	Canon
Lee Farrell	Canon
Wouter van der Beek	Cisco
William Lupton	Conexant
Grzegorz Kafel	Comarch
Shinichi Tsuruyama	Epson
Shivaun Albright	HP
John Ritchie	Intel
Mark Walker	Intel
Colleen Evans	Microsoft
Henry Rawas	Microsoft
Toby Nixon	Microsoft
Trevor Freeman	Microsoft
Cathy Chan	Nokia
Franklin Reynolds	Nokia
Jose Costa-Requena	Nokia
Yinghua Ye	Nokia
Geert Knapen	Philips
Jarno Guidi	Philips
Lex Heerink	Philips
Tom McGee	Philips
Andrew Fiddian-Green	Siemens
Markus Wischy	Siemens
John Fuller	Sony

<b>Authors of Annex A*</b>	<b>Company</b>
Chris Grundeman	CableLabs
Bich Nguyen	Cisco
Barbara Stark	AT&T
Clarke Stevens	CableLabs

<b>Authors of Annex C*</b>	<b>Company</b>
Clarke Stevens	Cable Labs Inc
Wouter van der Beek	Cisco Systems
Keith Miller (Chair)	Intel
Jeffrey Kang	TP Vision
Mateusz Belicki	Comarch

<b>Contributors of Annex C*</b>	<b>Company</b>
Clarke Stevens	Cable Labs Inc
Wouter van der Beek	Cisco Systems
Peter Waher	Clayster
Bich Nguyen	GoPro
Keith Miller	Intel
Jeffrey Kang	TP Vision
Mateusz Belicki	Comarch

**\*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

Introduction.....	9
0 Addressing .....	15
0.1 Determining whether to use Auto-IP .....	16
0.2 Choosing an address.....	16
0.3 Testing the address.....	16
0.4 Forwarding rules .....	17
0.5 Periodic checking for dynamic address availability .....	17
0.6 Device naming and DNS interaction .....	17
0.7 Name to IP address resolution.....	18
0.8 References.....	18
1 Discovery .....	18
1.1 SSDP message format .....	21
1.1.1 SSDP Start-line .....	21
1.1.2 SSDP message header fields .....	21
1.1.3 SSDP header field extensions .....	22
1.1.4 UUID format and recommended generation algorithms .....	22
1.1.5 SSDP processing rules .....	22
1.2 Advertisement .....	23
1.2.1 Advertisement protocols and standards .....	23
1.2.2 Device available - NOTIFY with ssdp:alive.....	24
1.2.3 Device unavailable -- NOTIFY with ssdp:byebye.....	31
1.2.4 Device Update – NOTIFY with ssdp:update .....	32
1.3 Search .....	34
1.3.1 Search protocols and standards .....	35
1.3.2 Search request with M-SEARCH.....	35
1.3.3 Search response .....	39
1.4 References.....	42
2 Description .....	42
2.1 Generic requirements on HTTP usage .....	45
2.2 Generic requirements on XML usage.....	48
2.3 Device description .....	48
2.4 UPnP Device Template .....	54
2.5 Service description .....	54
2.5.1 Defining and processing extended data types .....	61
2.5.2 String equivalents of extended data types .....	63
2.5.3 Generic requirements .....	64
2.5.4 Ordering of Elements.....	64
2.5.5 Versioning .....	64
2.6 UPnP Service Template .....	65
2.7 Non-standard vendor extensions and limitations.....	65
2.7.1 Placement of Additional Elements and Attributes .....	66
2.8 UPnP Device Schema .....	67
2.9 UPnP Service Schema .....	67

2.10	UPnP Datatype Schema .....	67
2.11	Retrieving a description using HTTP .....	67
2.12	References .....	70
3	Control .....	71
3.1	Control protocols .....	73
3.1.1	SOAP Profile .....	73
3.2	Actions .....	77
3.2.1	Action invocation .....	77
3.2.2	Action Response .....	80
3.2.3	UPnP Action Schema .....	82
3.2.4	Recommendations and additional requirements .....	83
3.2.5	Action error response .....	83
3.2.6	UPnP Error Schema .....	86
3.3	Query for variable .....	87
3.4	References .....	87
4	Eventing .....	87
4.1	Unicast eventing .....	88
4.1.1	Subscription .....	89
4.1.2	SUBSCRIBE with NT and CALLBACK .....	91
4.1.3	Renewing a subscription with SUBSCRIBE with SID .....	94
4.1.4	Canceling a subscription with UNSUBSCRIBE .....	95
4.2	Multicast Eventing .....	97
4.3	Event messages .....	98
4.3.1	Error Cases .....	99
4.3.2	Unicast eventing: Event messages: NOTIFY .....	99
4.3.3	Multicast Eventing: Event messages: NOTIFY .....	103
4.4	UPnP Event Schema .....	106
4.5	Augmenting the UPnP Device and Service Schemas .....	106
4.6	References .....	106
5	Presentation .....	107
5.1	References .....	108
Annex A	(normative) IP Version 6 Support .....	109
A.0	Note (informative) .....	109
A.1	Introduction .....	109
A.2	General Principles .....	110
A.2.1	UPnP Device Architecture V1.0 .....	110
A.2.2	UPnP Device Architecture V2.0 .....	110
A.2.3	IPv6 and Dual Stack .....	110
A.2.4	Device operation .....	112
A.2.5	Control point operation .....	112
A.3	Addressing .....	112
A.3.1	UPnP Messaging on IPv6 Interfaces .....	113
A.3.2	Summary of boot/startup process .....	113
A.3.3	Address Selection and RFC 6724 .....	113
A.4	Discovery .....	113
A.4.1	OPT and NLS .....	114

A.4.2	Advertisement .....	114
A.4.3	Advertisement: Device unavailable .....	115
A.4.4	Advertisement: Device update .....	115
A.4.5	Search.....	115
A.4.6	Search response .....	116
A.5	Description .....	116
A.6	Control .....	116
A.7	Eventing.....	116
A.8	Presentation.....	117
A.9	References.....	117
A.9.1	Normative.....	117
A.9.2	Informative .....	118
Annex B	Schemas .....	119
B.1	UPnP Device Schema .....	119
B.2	UPnP Service Schema .....	123
B.3	UPnP Control Schema.....	128
B.4	UPnP Error Schema .....	129
B.5	UPnP Event Schema .....	130
B.6	UPnP Cloud Schema.....	130
B.7	Schema references .....	132
Annex C	Cloud.....	133
C.1	Introduction .....	133
C.1.1	What is UPnP™ Cloud Technology (UCA)? .....	133
C.1.2	Audience .....	133
C.1.3	In this Annex .....	133
C.1.4	UDA compared to UCA.....	135
C.1.5	UCA General Communications Paths .....	137
C.1.6	UCA Specific Communication Paths .....	138
C.1.7	UCA Steps as Analogies to UDA .....	139
C.2	Terms and Definitions .....	141
C.2.1	Acronyms .....	141
C.2.2	General Cloud Terms and Definitions .....	141
C.2.3	Device and Control Point Terms and Definitions .....	142
C.2.4	Service Terms and Definitions .....	142
C.2.5	Groups .....	142
C.3	References.....	143
C.4	General XMPP Features.....	144
C.4.1	XMPP Jabber IDs or <a href="#">JIDs</a> .....	144
C.5	Creating a Device or Control Point Resource.....	145
C.5.1	Finding a UCS .....	145
C.5.2	Account Creation .....	146
C.5.3	Authentication .....	146
C.5.4	Binding Devices and Control Points as a Resource.....	149
C.5.5	Embedded Devices.....	152
C.6	Presence and Discovery.....	154
C.6.1	Presence (Analog to NOTIFY with ssdp:alive).....	154

C.6.2	XMPP disco#items (analog to M-SEARCH for users UCCDs and UCC-CPs) .....	158
C.6.3	Presence update (analog to NOTIFY with sdp:update) .....	159
C.6.4	Presence "unavailable" (Analog to NOTIFY with sdp:byebye) .....	160
C.6.5	Service Level Discovery .....	160
C.6.6	IQ:Query for DDD and SCPD Exchange (analog of HTTP GET for DDD and SCPD) .....	160
C.7	<a href="#">PubSub</a> (Analog of Eventing) .....	169
C.7.1	Creating the UCCD <a href="#">PubSub</a> structure .....	173
C.7.2	Creating a UCCD PubSub collection .....	176
C.7.3	Publishing a UCCD PubSub event .....	180
C.7.4	Subscribing to a UCCD PubSub collection .....	184
C.7.5	Unsubscribing to a UCCD <a href="#">PubSub</a> collection .....	186
C.7.6	Permissions model .....	187
C.8	SOAP over XMPP (Analog of Control) .....	188
C.9	Support for Binary (Media) Transport .....	192
C.10	UCA errorCodes .....	192
C.11	UCA Schemas .....	192
C.12	Closing a UCA Session .....	193
C.13	UCA over BOSH and WebSocket .....	193
Figure 1:	— Protocol stack .....	10
Figure 1-1:	— Discovery architecture .....	19
Figure 1-2:	— Advertisement protocol stack .....	23
Figure 1-3:	— Initial and repeat announcements, no announcement spreading .....	26
Figure 1-4:	— Initial and repeat announcements, message spreading of repeat announcements .....	27
Figure 1-5:	— Search protocol stack .....	35
Figure 2-1:	— Description architecture .....	42
Figure 2-2:	— Description retrieval protocol stack .....	68
Figure 3-1:	— Control architecture .....	71
Figure 3-2:	— Control protocol stack .....	73
Figure 4-1:	— Unicast eventing architecture .....	88
Figure 4-2:	— Unicast eventing protocol stack .....	89
Figure 4-3:	— Multicast eventing architecture .....	97
Figure 4-4:	— Multicast eventing protocol stack .....	98
Figure 5-1:	— Presentation architecture .....	107
Figure 5-2:	— Presentation protocol stack .....	107
Figure C-1:	— Protocol stacks UDA versus UCA .....	135
Figure C-2:	— Protocol stack UCA UCCD/UCC-CP and UCA Servers (UCS or UCOD) .....	136
Figure C-3:	— General UCA Configuration .....	138
Figure C-4:	— Specific UCA communications .....	139
Figure C-5:	— XMPP Authentication Negotiation .....	146
Figure C-6:	— Stanza routing for applications with UCA and other XMPP functionality .....	151



Figure C-7: — UDA to UCA Mapping of embedded devices.....	154
The individual <a href="#">presence</a> exchange between the UCCDs, UCC-CPs, and UCS for an N connected UPnP scenario is illustrated in .....	158
Figure C-8: — Self <presence> stanza flows.....	158
Figure C-9: — Combined Connect, Announce and Describe Message Flow .....	167
Figure C-10: — <a href="#">PubSub</a> Hierarchy Event Structure Creation.....	173
Figure C-11: — BOSH and WebSocket UCA Stack .....	193
Figure C-12: — BOSH and WebSocket at UCA component stacks .....	195
Table 1 — Acronyms.....	13
Table 1-1 — Root device discovery messages .....	24
Table 1-2 — Embedded device discovery messages .....	24
Table 1-3 — Service discovery messages .....	25
Table 2-1: — Vendor extensions .....	65
Table 3-1: — SOAP 1.1 UPnP Profile.....	74
Table 3-2: — <a href="#">mustUnderstand</a> attribute.....	75
Table 3-3: — UPnP Defined Action error codes.....	85
Table 4-4: — HTTP Status Codes indicating a Subscription Error .....	94
Table 4-5: — HTTP Status Codes indicating a Resubscription Error .....	95
Table 4-6: — HTTP Status Codes indicating a Cancel Subscription Error .....	97
Table 4-7: — HTTP Status Codes indicating a Notify Error .....	102
Table 4-8: — Multicast event levels .....	104
Table A-1: — Matching of Device Address to Multicast Scope.....	112
Table C-1: — Acronyms .....	141
Table C-2: — Mapping of DDD <a href="#">iconList</a> to [XEP-0084].....	164
Table C-3: — Summary of Requirements for DDD elements.....	168
Table C-4: — <a href="#">PubSub Node</a> Types .....	170
Table C-5: — <a href="#">PubSub</a> Node Access Models .....	170
Table C-6: — PubSub Affiliations and their Privileges to "publishing" as defined by [XEP-0060] and further restricted by UCA (see footnotes).....	171
Table C-7: — <a href="#">PubSub</a> Affiliations and their Privileges to "subscribers".....	171

## Introduction

### What is UPnP<sup>1</sup> Technology?

UPnP technology defines an architecture for pervasive peer-to-peer network connectivity of intelligent appliances, wireless devices, and PCs of all form factors. It is designed to bring easy-to-use, flexible, standards-based connectivity to ad-hoc or unmanaged networks whether in the home, in a small business, public spaces, or attached to the Internet. UPnP technology provides a distributed, open networking architecture that leverages TCP/IP and Web technologies to enable seamless proximity networking in addition to control and data transfer among networked devices.

The UPnP Device Architecture (UDA) is more than just a simple extension of the plug and play peripheral model. It is designed to support zero-configuration, "invisible" networking, and automatic discovery for a breadth of device categories from a wide range of vendors. This means a device can dynamically join a network, obtain an IP address, convey its capabilities, and learn about the presence and capabilities of other devices. Finally, a device can leave a network smoothly and automatically without leaving any unwanted state behind.

The technologies leveraged in the UPnP architecture include Internet protocols such as IP, TCP, UDP, HTTP, and XML. Like the Internet, contracts are based on wire protocols that are declarative, expressed in XML, and communicated via HTTP. Using Internet protocols is a strong choice for UDA because of its proven ability to span different physical media, to enable real world multiple-vendor interoperation, and to achieve synergy with the Internet and many home and office intranets. The UPnP architecture has been explicitly designed to accommodate these environments. Further, via bridging, UDA accommodates media running non-IP protocols when cost, technology, or legacy prevents the media or devices attached to it from running IP.

What is "universal" about UPnP technology? No device drivers; common protocols are used instead. UPnP networking is media independent. UPnP devices can be implemented using any programming language, and on any operating system. The UPnP architecture does not specify or constrain the design of an API for applications; OS vendors may create APIs that suit their customers' needs.

### UPnP Forum

UPnP Forum is an industry initiative designed to enable easy and robust connectivity among stand-alone devices and PCs from many different vendors. UPnP Forum seeks to develop standards for describing device protocols and XML-based device schemas for the purpose of enabling device-to-device interoperability in a scalable, networked environment.

UPnP Forum is comprised of member companies across many industries that promote the adoption of uniform technical device interconnectivity standards and testing and certifying of these devices. The Forum develops and administers the testing and certification process, administers the UPnP logo program, and provides information to members and other interested parties regarding the certification of UPnP devices. The UPnP device certification process is open to any vendor who is an implementer level member of UPnP Forum, has paid the implementer dues, and has devices that support UPnP functionality. For more information, see <http://www.upnp.org>.

UPnP Forum has set up working committees in specific areas of domain expertise. These working committees are charged with creating proposed device standards, building sample implementations, and building appropriate test suites. This document indicates specific technical decisions that are the purview of UPnP Forum working committees.

UPnP vendors can build compliant devices with confidence of interoperability and benefits of shared intellectual property and the logo program. Separate from the logo program, vendors

---

<sup>1</sup> The UPnP® Word Mark and UPnP® Logo are certification marks owned by UPnP Forum.

may also build devices that adhere to the UPnP Device Architecture defined herein without a formal standards procedure. If vendors build non-standard devices, they determine technical decisions that would otherwise be determined by a UPnP Forum working committee.

### In this document

The UPnP Device Architecture (formerly known as the DCP Framework) contained herein defines the protocols for communication between controllers, or *control points*, and devices. For discovery, description, control, eventing, and presentation, the UPnP Device Architecture uses the following protocol stack (the indicated colors and type styles are used throughout this document to indicate where each protocol element is defined):

**Figure 1: — Protocol stack**

<i>UPnP vendor</i> [purple-italic]			
<i>UPnP Forum</i> [red-italic]			
<b>UPnP Device Architecture</b> [green-bold]			
<u>SSDP</u> [blue]	<b>Multicast events</b> [navy-bold]	<u>SOAP</u> [blue]	<b>GENA</b> [navy-bold]
		HTTP [black]	HTTP [black]
UDP [black]		TCP [black]	
IP [black]			

At the highest layer, messages logically contain only UPnP vendor-specific information about their devices. Moving down the stack, vendor content is supplemented by information defined by UPnP Forum working committees. Messages from the layers above are hosted in UPnP-specific protocols such as the Simple Service Discovery Protocol (SSDP), the General Event Notification Architecture (GENA) and the multicast event protocol defined in this document, and others that are referenced. SSDP is delivered via either multicast or unicast UDP. Multicast events are delivered via multicast UDP. **GENA** is delivered via HTTP. Ultimately, all messages above are delivered over IP. The remaining clauses of this document describe the content and format for each of these protocol layers in detail. For reference, colors in [square brackets] above indicate which protocol defines specific message components throughout this document.

Two general classifications of devices are defined by the UPnP architecture: controlled devices (or simply “devices”), and control points. A controlled device functions in the role of a server, responding to requests from control points. Both control points and controlled devices can be implemented on a variety of platforms including personal computers and embedded systems. Multiple devices, control points, or both may be operational on the same network endpoint simultaneously.

Note: This document is oriented toward an IPv4 environment. Considerations for an IPv6 environment are expressed in Annex A.

The foundation for UPnP networking is IP addressing. In an IPv4 environment, each device or control point shall have a Dynamic Host Configuration Protocol (DHCP) client and search for a DHCP server when the device or control point is first connected to the network. If a DHCP server is available, i.e., the network is managed; the device or control point shall use the IP address assigned to it. If no DHCP server is available, i.e., the network is unmanaged; the device or control point shall use Auto IP to get an address. In brief, Auto IP defines how a device or control point intelligently chooses an IP address from a set of reserved addresses and is able to move easily between managed and unmanaged networks. If during the DHCP transaction, the device or control point obtains a domain name, e.g., through a DNS server or via DNS forwarding, the device or control point should use that name in subsequent network operations; otherwise, the device or control point should use its IP address.

Certain UPnP networks have more complex configurations such as multiple physical networks and/or multiple logical networks to accommodate multiple non-overlapping addressing schemes. Devices and control points may also have two or more network interfaces, and/or two or more IP addresses assigned to each interface. In such configurations, a single device or control point may be assigned multiple IP addresses from different logical networks in the same UPnP network, resulting in devices appearing to a control point multiple times in the network. Devices and control points that have multiple IP addresses on the same UPnP network are referred to as multi-homed. Throughout this document, the term "UPnP-enabled interface" is used to refer to an interface which is assigned an IP address belonging to the UPnP network. Additional behaviors specific to multi-homed devices and control points will be covered in applicable clauses throughout the document. However, as a general principle, related interactions between control points and devices (e.g. action control request and response messages, event subscription and event messages) shall occur using the same pair of outgoing and incoming UPnP-enabled interfaces.

Given an IP address, Step 1 in UPnP networking is *discovery*. When a device is added to the network, the UPnP discovery protocol allows that device to advertise its services to control points on the network. Similarly, when a control point is added to the network, the UPnP discovery protocol allows that control point to search for devices of interest on the network. The fundamental exchange in both cases is a discovery message containing a few essential specifics about the device or one of its services, e.g., its type, identifier, and a pointer to more detailed information. The clause on Discovery below explains how devices advertise, how control points search, and contains details about the format of discovery messages.

Step 2 in UPnP networking is *description*. After a control point has discovered a device, the control point still knows very little about the device. For the control point to learn more about the device and its capabilities, or to interact with the device, the control point shall retrieve the device's description from the URL provided by the device in the discovery message. Devices may contain other logical devices, as well as functional units, or *services*. The UPnP description for a device is expressed in XML and includes vendor-specific manufacturer information like the model name and number, the serial number, the manufacturer name, URLs to vendor-specific Web sites, etc. The description also includes a list of any embedded devices or services, as well as URLs for control, eventing, and presentation. For each service, the description includes a list of the commands, or *actions*, to which the service responds, and parameters, or *arguments* for each action; the description for a service also includes a list of variables; these variables model the state of the service at run time, and are described in terms of their data type, range, and event characteristics. The clause on Description below explains how devices are described and how control points retrieve those descriptions.

Step 3 in UPnP networking is *control*. After a control point has retrieved a description of the device, the control point can send actions to a device's services. To do this, a control point sends a suitable control message to the control URL for the service (provided in the device description). Control messages are also expressed in XML using the Simple Object Access Protocol (SOAP). Like function calls, in response to the control message, the service returns any action-specific values. The effects of the action, if any, are modeled by changes in the variables that describe the run-time state of the service. The clause on Control below explains the description of actions, state variables, and the format of control messages.

Step 4 in UPnP networking is *eventing*. A UPnP description for a service includes a list of actions the service responds to and a list of variables that model the state of the service at run time. The service publishes updates when these variables change, and a control point may subscribe to receive this information. The service publishes updates by sending event messages. Event messages contain the names of one or more state variables and the current value of those variables. These messages are also expressed in XML. A special initial event message is sent when a control point first subscribes; this event message contains the names and values for all evented variables and allows the subscriber to initialize its model of the state of the service. To support scenarios with multiple control points, eventing is designed to keep all control points equally informed about the effects of any action. Therefore, all subscribers are sent all event messages, subscribers receive event messages for all evented

variables that have changed, and event messages are sent no matter why the state variable changed (either in response to a requested action or because the state the service is modeling changed). Multicast eventing is a variant of Step 4 in UPnP networking. Through multicast eventing, control points can listen to state changes in services without subscription. This form of eventing is useful first when events which are not relevant to specific UPnP interactions should be delivered to control points to inform users, and second when multiple controlled devices want to inform multiple other control points. Multicast eventing is inherently unreliable since it is based on UDP. To increase the probability of successful transmission, the option to retransmit multicast event notifications is outlined. UPnP Working committees should define whether specific events are multicast events. The clause on Eventing below explains unicast event subscription and the format of both unicast and multicast event messages.

Step 5 in UPnP networking is *presentation*. If a device has a URL for presentation, then the control point can retrieve a page from this URL, load the page into a browser, and depending on the capabilities of the page, allow a user to control the device and/or view device status. The degree to which each of these can be accomplished depends on the specific capabilities of the presentation page and device. The clause on Presentation below explains the protocol for retrieving a presentation page.

## **Audience**

The audience for this document includes UPnP device and control point vendors, members of UPnP Forum working committees, and anyone else who has a need to understanding the technical details of UPnP protocols.

This document assumes the reader is familiar with the HTTP, TCP, UDP, IP family of protocols; this document makes no attempt to explain them. This document also assumes most readers will be new to XML, and while it is not an XML tutorial, XML-related issues are addressed in detail given the centrality of XML to the UPnP Device Architecture. This document makes no assumptions about the reader's understanding of various programming or scripting languages.

## **Conformance terminology**

In this document, features are described as required, recommended, allowed or DEPRECATED as follows:

Required (or shall or mandatory).

These basic features shall be implemented to comply with UPnP Device Architecture. The phrases "shall not", and "PROHIBITED" indicate behavior that is prohibited, i.e. that if performed means the implementation is not in compliance.

Recommended (or should).

These features add functionality supported by UPnP Device Architecture and should be implemented. Recommended features take advantage of the capabilities UPnP Device Architecture, usually without imposing major cost increases. Notice that for compliance testing, if a recommended feature is implemented, it shall meet the specified requirements to be in compliance with these guidelines. Some recommended features could become requirements in the future. The phrase "should not" indicates behavior that is permitted but not recommended.

Allowed (or may).

These features are neither required nor recommended by UPnP Device Architecture, but if the feature is implemented, it shall meet the specified requirements to be in compliance with these guidelines. These features are not likely to become requirements in the future.

DEPRECATED.

Although these features are still described in this specification, they should not be implemented except for backward compatibility. The occurrence of a deprecated feature during operation of an implementation compliant with the current specification has no effect on the implementation's operation and does not produce any error conditions. Backward compatibility may require that a feature is implemented and functions as specified but it shall never be used by implementations compliant with this specification.

## Acronyms

**Table 1 — Acronyms**

Acronym	Meaning	Acronym	Meaning
ARP	Address Resolution Protocol	SOAP	Simple Object Access Protocol
CP	Control Point	SSDP	Simple Service Discovery Protocol
DCP	Device Control Protocol	UDA	UPnP Device Architecture
DDD	Device Description Document	UPC	Universal Product Code
DHCP	Dynamic Host Configuration Protocol	URI	Uniform Resource Identifier
DNS	Domain Name System	URL	Uniform Resource Locator
GENA	General Event Notification Architecture	URN	Uniform Resource Name
HTML	Hypertext Markup Language	UUID	Universally Unique Identifier
HTTP	Hypertext Transfer Protocol	XML	Extensible Markup Language
SCPD	Service Control Protocol Description		

## Glossary

### action

Command exposed by a service. Takes one or more input or output arguments. May have a return value. For more information, see clause 2, "Description" and clause 3, "Control".

### argument

Parameter for action exposed by a service. May be in or out. For more information, see clause 2, "Description" and clause 3, "Control".

### control point

Retrieves device and service descriptions, sends actions to services, polls for service state variables, and receives events from services.

### device

Logical device. A container. May embed other logical devices. Embeds one or more services. Advertises its presence on network(s). For more information, see clause 1, "Discovery" and clause 2, "Description".

### device description

Formal definition of a logical device, expressed in the UPnP Template Language. Written in XML syntax. Specified by a UPnP vendor by filling in the placeholders in a UPnP Device Template, including, e.g., manufacturer name, model name, model number, serial number, and URLs for control, eventing, and presentation. For more information, see clause 2, "Description".

### device type

Standard device types are denoted by urn:schemas-upnp-org:device: followed by a unique name assigned by a UPnP Forum working committee. One-to-one relationship with UPnP Device Templates. UPnP vendors may specify additional device types; these are denoted by urn:domain-name:device: followed by a unique name assigned by the vendor, where *domain-name* is a Vendor Domain Name. For more information, see clause 2, "Description".

**event**

Notification of one or more changes in state variables exposed by a service. For more information, see clause 4, “Eventing”.

**GENA**

General Event Notification Architecture. The event subscription and notification protocol defined in clause 4, “Eventing”.

**publisher**

Source of event messages. Typically a device's service. For more information, see clause 4, “Eventing”.

**root device**

A logical device that is not embedded in any other logical device. For more information, see clause 2, “Description”.

**service**

Logical functional unit. Smallest units of control. Exposes actions and models the state of a physical device with state variables. For more information, see clause 3, “Control”.

**service description**

Formal definition of a logical service, expressed in the UPnP Template language. Written in XML syntax. Specified by a UPnP vendor by filling in any placeholders in a UPnP Service Template. (Was SCPD.) For more information, see clause 2, “Description”.

**service type**

Standard service types are denoted by urn:schemas-upnp-org:service: followed by a unique name assigned by a UPnP forum working committee, colon, and an integer version number. One-to-one relationship with UPnP Service Templates. UPnP vendors may specify additional services; these are denoted by urn:*domain-name*:service: followed by a unique name assigned by the vendor, colon, and a version number, where *domain-name* is a Vendor Domain Name. For more information, see clause 2, “Description”.

**SOAP**

Simple Object Access Protocol. A remote-procedure call mechanism based on XML that sends commands and receives values over HTTP. For more information, see clause 3, “Control”.

**SSDP**

Simple Service Discovery Protocol. A multicast discovery and search mechanism that uses a multicast variant of HTTP over UDP. Defined in clause 1, “Discovery”.

**state variable**

Single facet of a model of a physical service. Exposed by a service. Has a name, data type, optional default value, optional constraints values, and may trigger events when its value changes. For more information, see clause 2, “Description” and clause 3, “Control”.

**subscriber**

Recipient of event messages. Typically a control point. For more information, see clause 4, “Eventing”.

**UPnP Device Template**

Template listing device type, required embedded devices (if any), and required services. Written in XML syntax and derived from the UPnP Device Schema. Defined by a UPnP Forum working committee. One-to-one relationship with standard device types. For more information, see clause 2, “Description”.

**UPnP Service Template**

Template listing action names, parameters for those actions, state variables, and properties of those state variables. Written in XML syntax and derived from the UPnP Service Schema.

Defined by a UPnP Forum working committee. One-to-one relationship with standard service types. For more information, see clause 2, “Description”.

### **UPnP Device Schema**

Defines the elements and attributes used in UPnP Device and Service Templates. Written in XML syntax and derived from XML Schema (Part 1: Structures, Part 2: Datatypes). Defined by the UPnP Device Architecture herein. For more information, see clause 2, “Description”.

### **Vendor Domain Name**

A domain name that is supplied by a vendor. It is owned by the vendor, and shall be registered with an ICANN accredited Registrar, such that it is unique. The vendor shall keep the domain name registration up to date. A Vendor Domain Name length should be chosen to be compatible with the use of the domain name in the UDA.

### **References and resources**

RFC 2710, Multicast Listener Discovery for IPv6. Available at:  
<http://www.ietf.org/rfc/rfc2710.txt>.

RFC 2616, HTTP: Hypertext Transfer Protocol 1.1. Available at:  
<http://www.ietf.org/rfc/rfc2616.txt>.

RFC 2279, UTF-8, a transformation format of ISO 10646 (character encoding). Available at:  
<http://www.ietf.org/rfc/rfc2279.txt>.

XML, Extensible Markup Language. W3C recommendation. Available at:  
<http://www.w3.org/XML/>.

DEVICEPROTECTION, UPnP Device Protection specification. Available at  
<http://upnp.org/specs/gw/UPnP-gw-DeviceProtection-v1-Service.pdf>.

Each clause in this document contains additional information about resources for specific topics.

## **0 Addressing**

*Addressing is Step 0 of UPnP networking. Through addressing, devices and control points get a network address. Addressing enables discovery (Step 1) where control points find interesting device(s), description (Step 2) where control points learn about device capabilities, control (Step 3) where a control point sends commands to device(s), eventing (Step 4) where control points listen to state changes in device(s), and presentation (Step 5) where control points display a user interface for device(s).*

The foundation for UPnP networking is IP addressing. A UPnP device or control point is allowed to support IP version 4-only, or both IP version 4 and IP version 6. This clause, and the examples given throughout clauses 1 through 5 of this document, assumes an IPv4 implementation. Annex A of this document describes IPv6 operation. Each UPnP device or control point which does not itself implement a DHCP server shall have a Dynamic Host Configuration Protocol (DHCP) client and search for a DHCP server when the device or control point is first connected to the network (if the device or control point itself implements a DHCP server, it allowed to allocate itself an address from the pool that it controls). If a DHCP server is available, i.e., the network is managed; the device or control point shall use the IP address assigned to it. If no DHCP server is available, i.e., the network is unmanaged; the device or control point shall use automatic IP addressing (Auto-IP) to obtain an address.

Auto-IP (defined in RFC 3927) defines how a device or control point: (a) determines if DHCP is unavailable, and (b) intelligently chooses an IP address from a set of link-local IP addresses. This method of address assignment enables a device or control point to easily move between managed and unmanaged networks.



This clause provides an overview of the basic operation of Auto-IP. The operations described in this clause are detailed and clarified in the reference documents listed below. Where conflicts between this document and the reference documents exist, the reference document always takes precedence.

### **0.1 Determining whether to use Auto-IP**

A device or control point that supports Auto-IP and is configured for dynamic address assignment begins by requesting an IP address via DHCP by sending out a DHCPDISCOVER message. The amount of time this DHCP Client listens for DHCPOFFERS is implementation dependent. If a DHCPOFFER is received during this time, the device or control point shall continue the process of dynamic address assignment. If no valid DHCPOFFERS are received, the device or control point shall then auto-configure an IP address using Auto-IP.

### **0.2 Choosing an address**

To auto-configure an IP address using Auto-IP, the device or control point uses an implementation dependent algorithm for choosing an address in the 169.254/16 range. The first and last 256 addresses in this range are reserved and shall NOT be used.

The selected address shall then be tested to determine if the address is already in use. If the address is in use by another device or control point, another address shall be chosen and tested, up to an implementation dependent number of retries. The address selection shall be randomized to avoid collision when multiple devices or control points are attempting to allocate addresses. The device or control point chooses an address using a pseudo-random algorithm (distributed over the entire address range from 169.254.1.0 to 169.254.254.255) to minimize the likelihood that devices or control points that join the network at the same time will choose the same address and subsequently choose alternative addresses in the same sequence when collisions are detected. This pseudo-random algorithm should be seeded using the device's or control point's Ethernet hardware MAC address.

### **0.3 Testing the address**

To test the chosen address, the device or control point shall use an Address Resolution Protocol (ARP) probe. An ARP probe is an ARP request with the device or control point hardware address used as the sender's hardware address and the sender's IP address set to 0s. The device or control point shall then listen for responses to the ARP probe, or other ARP probes for the same IP address. If either of these ARP packets is seen, the device or control point shall consider the address in use and try a different address. The ARP probe is allowed to be repeated for greater certainty that the address is not already in use; it is recommended that the probe be sent four times at two-second intervals.

After successfully configuring a link-local address, the device or control point shall send two gratuitous ARPs, spaced two seconds apart, this time filling in the sender IP address. The purpose of these gratuitous ARPs is to make sure that other hosts on the net do not have stale ARP cache entries left over from some other host that may previously have been using the same address.

Devices and control points that are equipped with persistent storage are allowed to record the IP address they have selected and on the next boot use that address as their first candidate when probing, in order to increase the stability of addresses and reduce the need to resolve address conflicts.

Address collision detection is not limited to the address testing phase, when the device or control point is sending ARP probes and listening for replies. Address collision detection is an ongoing process that is in effect for as long as the device or control point is using a link-local address. At any time, if a device or control point receives an ARP packet with its own IP address given as the sender IP address, but a sender hardware address that does not match its own hardware address, then the device or control point shall treat this as an address collision and shall respond as described in either a) or b) below:

- a) Immediately configure a new link-local IP address as described above; or,
- b) If the device or control point currently has active TCP connections or other reasons to prefer to keep the same IP address, and has not seen any other conflicting ARP packets recently (e.g., within the last ten seconds) then it is allowed to elect to attempt to defend its address once, by recording the time that the conflicting ARP packet was received, and then broadcasting one single gratuitous ARP, giving its own IP and hardware addresses as the source addresses of the ARP. However, if another conflicting ARP packet is received within a short time after that (e.g., within ten seconds) then the device or control point shall immediately configure a new Auto-IP address as described above.

The device or control point shall respond to conflicting ARP packets as described in either a) or b) above; it shall NOT ignore conflicting ARP packets. If a new address is selected, the device or control point shall cancel previous advertisements and re-advertise with the new address.

After successfully configuring an Auto-IP address, all subsequent ARP packets (replies as well as requests) containing an Auto-IP source address shall be sent using link-level *broadcast* instead of link-level *unicast*, in order to facilitate timely detection of duplicate addresses.

#### **0.4 Forwarding rules**

IP packets whose source or destination addresses are in the 169.254/16 range shall NOT be sent to any router for forwarding. Instead, the senders shall ARP for the destination address and then send the packets directly to the destination on the same link. IP datagrams with a multicast destination address and an Auto-IP source address shall NOT be forwarded off the local link. Devices and control points are allowed to assume that all 169.254/16 destination addresses are on-link and directly reachable. The 169.254/16 address range shall not be subnetted.

#### **0.5 Periodic checking for dynamic address availability**

A device or control point that has auto-configured an IP address shall periodically check for the existence of a DHCP server. This is accomplished by sending DHCPDISCOVER messages. How often this check is made is implementation dependent, but checking every 5 minutes would maintain a balance between network bandwidth required and connectivity maintenance. If a DHCPOFFER is received, the device or control point shall proceed with dynamic address allocation. Once a DHCP assigned address is in place, the device or control point is allowed to release the auto-configured address, but is also allowed to choose to maintain this address for a period of time (or indefinitely) to maintain connectivity.

To switch over from one IP address to a new one, the device should, if possible, cancel any outstanding advertisements made on the previous address and shall issue new advertisements on the new address. The clause on Discovery explains advertisements and their cancellations. In addition, any event subscriptions are deleted by the device (see clause on Eventing).

For a multi-homed device with multiple IP addresses, to switch one of the IP addresses to a new one, the device should cancel any outstanding advertisements made on the previous IP address, and shall issue new advertisements on the new IP addresses. Furthermore, it shall also issue appropriate update advertisements on all unaffected IP addresses. The clause on Discovery explains advertisements, their cancellations and updates. The clause on Eventing explains the effect on event subscriptions.

#### **0.6 Device naming and DNS interaction**

Once a device has a valid IP address for the network, it can be located and referenced on that network through that address. There may be situations where the end user needs to locate and identify a device. In these situations, a friendly name for the device is much easier for a human to use than an IP address. If a device chooses to provide a host name to a DHCP

server and register with a DNS server, the device should either ensure the requested host name is unique or provide a means for the user to change the requested host name. Most often, devices do not provide a host name, but provide URLs using literal (numeric) IP addresses.

Moreover, names are much more static than IP addresses. Clients referring a device by name don't require any modification when the IP address of a device changes. Mapping of the device's DNS name to its IP address could be entered into the DNS database manually or dynamically according to RFC 2136. While devices supporting dynamic DNS updates can register their DNS records directly in the DNS, it is also possible to configure a DHCP server to register DNS records on behalf of these DHCP clients.

## 0.7 Name to IP address resolution

A device that needs to contact another device identified by a DNS name needs to discover its IP address. The device submits a DNS query according to RFC1034 and 1035 to the pre-configured DNS server(s) and receives a response from a DNS server containing the IP address of the target device. A device can be statically pre-configured with the list of DNS servers. Alternatively a device could be configured with the list of DNS server through DHCP, or after the address assignment through a DHCPINFORM message.

## 0.8 References

RFC1034, Domain Names - Concepts and Facilities. Available at:  
<http://www.ietf.org/rfc/rfc1034.txt>.

RFC1035, Domain Names - Implementation and Specification. Available at:  
<http://www.ietf.org/rfc/rfc1035.txt>.

RFC 2131, Dynamic Host Configuration Protocol. Available at:  
<http://www.ietf.org/rfc/rfc2131.txt>.

RFC 2136, Dynamic Updates in the Domain Name System. Available at:  
<http://www.ietf.org/rfc/rfc2136.txt>.

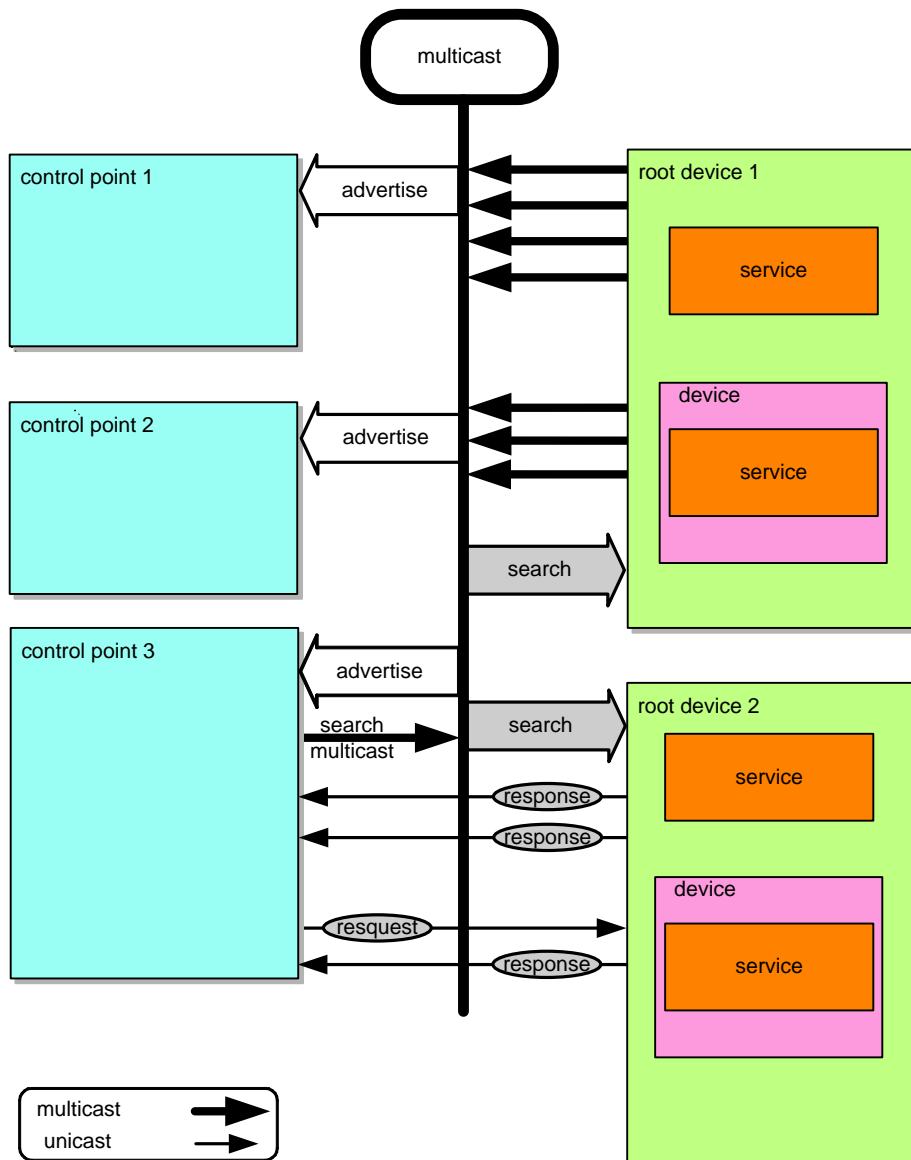
RFC 3927, Dynamic Configuration of IPv4 Link-Local Addresses. Available at:  
<http://www.ietf.org/rfc/rfc3927.txt>.

## 1 Discovery

*Discovery is Step 1 in UPnP™ networking. Discovery comes after addressing (Step 0) where devices get a network address. Through discovery, control points find interesting device(s). Discovery enables description (Step 2) where control points learn about device capabilities, control (Step 3) where a control point sends commands to device(s), eventing (Step 4) where control points listen to state changes in device(s), and presentation (Step 5) where control points display a user interface for device(s).*

Discovery is the first step in UPnP networking. When a device is added to the network, the UPnP discovery protocol allows that device to advertise its services to control points on the network. Similarly, when a control point is added to the network, the UPnP discovery protocol allows that control point to search for devices of interest on the network. The fundamental exchange in both cases is a discovery message containing a few, essential specifics about the device or one of its services, e.g., its type, universally unique identifier, a pointer to more detailed information and optionally parameters that identify the current state of the device.

Figure 1-1: — Discovery architecture



When a device knows it is newly added to the network, it shall multicast a number of discovery messages advertising itself, its embedded devices, and its services (initial announce). Any interested control point can listen to the standard multicast address for notifications that new capabilities are available. A multi-homed device shall multicast the discovery messages on all UPnP-enabled interfaces. A multi-homed control point is allowed to listen to the standard multicast address on one, some or all of its UPnP-enabled interfaces.

When a new control point is added to the network, it is allowed to multicast a discovery message searching for interesting devices, services, or both. All devices shall listen to the standard multicast address for these messages and shall respond if any of their root devices, embedded devices or services matches the search criteria in the discovery message. In addition, a control point is allowed to unicast a discovery message to a specific IP address on port 1900 or on the port specified by the optional SEARCHPORT.UPNP.ORG header field (which supersedes port 1900 for this use), searching for a UPnP device or service at that specific IP address. This action presumes the control point already knows the device at this IP address is a UPnP device (which listens on the appropriate port). The control point can use unicast search for a number of applications. A unicast search can quickly confirm a specific device and provide the corresponding discovery information (e.g. UUID, URL) of this device.

All devices shall listen to incoming unicast search messages on port 1900 or, if provided, the port number specified in the SEARCHPORT.UPNP.ORG header field and shall respond if any of their root devices, embedded devices or services matches the search criteria in the discovery message.

A multi-homed control point is allowed to multicast discovery messages on one, some or all of its UPnP-enabled interfaces. Multi-homed devices shall listen to the standard multicast address on all UPnP-enabled interfaces for multicast discovery messages. Multi-homed devices shall also listen to incoming unicast search messages on port 1900 or, if provided, the port number specified in the SEARCHPORT.UPNP.ORG header field. The devices shall respond if any of their root devices, embedded devices or services matches the search criteria in the discovery message.

To reiterate, a control point is allowed to learn of a device of interest because that device sent discovery messages advertising itself or because the device responded to a discovery message searching for devices. In either case, if a control point is interested in a device and wants to learn more about it, the control point uses the information in the discovery message to send a *description* query message. The clause on Description explains description messages in detail.

When a device is removed from the network, it should, if possible, multicast a number of discovery messages revoking its earlier announcements, effectively declaring that its root devices, embedded devices and services will no longer be available. When the IP address of a device is changed, it should revoke any earlier announcements and it shall advertise using the new IP address.

When a multi-homed device becomes unavailable to the network on any of its UPnP-enabled interfaces, it should, if possible, multicast a number of discovery messages revoking its earlier announcements on the affected UPnP-enabled interfaces, effectively declaring that its root devices, embedded devices and services will no longer be available on those interfaces. If it remains available to the network on any of its other UPnP-enabled interfaces, it shall NOT multicast such discovery messages on the unaffected UPnP-enabled interfaces.

When a multi-homed device becomes available to the network on a new UPnP-enabled interface (in addition to any existing UPnP-enabled interfaces), it shall increase its BOOTID.UPNP.ORG field value (see clause 1.2 “Advertisement”), and multicast a number of update messages on the existing UPnP-enabled interfaces to announce the new BOOTID.UPNP.ORG field value. After all the update messages have been sent, it shall multicast a number of discovery messages on all (existing and new) UPnP-enabled interfaces with the new BOOTID.UPNP.ORG field value.

Similarly, when one of the IP addresses of a multi-homed device is changed, it should revoke any earlier announcements on the previous IP address. It shall increase its BOOTID.UPNP.ORG field value (see clause 1.2 “Advertisement”), and multicast a number of update messages on the existing UPnP-enabled interfaces to announce the new BOOTID.UPNP.ORG field value. After all the update messages have been sent, it shall multicast a number of discovery messages on all (existing and new) UPnP-enabled interfaces with the new BOOTID.UPNP.ORG field value.

Finally, if a multi-homed device loses connectivity on one of its UPnP-enabled interfaces and then regains connectivity, it shall increase its BOOTID.UPNP.ORG field value (see 1.2, “Advertisement”), and multicast a number of update messages on the unaffected UPnP-enabled interfaces to announce the new BOOTID.UPNP.ORG field value. After all the update messages have been sent, it shall multicast a number of discovery messages on all (affected and unaffected) UPnP-enabled interfaces with the new BOOTID.UPNP.ORG field value.

To limit network congestion, the time-to-live (TTL) of each IP packet for each multicast message should default to 2 and should be configurable. When the TTL is greater than 1, it is possible for multicast messages to traverse multiple routers; therefore control points and devices using non-AutoIP addresses shall send an IGMP Join message so that routers will

forward multicast messages to them (this is not necessary when using an Auto-IP address, since packets with Auto-IP addresses will not be forwarded by routers).

**Versioning:** Discovery plays an important role in the interoperability of devices and control points using different versions of UPnP networking. The UPnP Device Architecture (defined herein) is versioned with both a major and a minor version, usually written as *major.minor*, where both *major* and *minor* are integers (for example, version 2.10 [two dot ten] is *newer* than version 2.2 [two dot two]). Advances in minor versions shall be a compatible superset of earlier minor versions of the same major version. Advances in major version are not required to be supersets of earlier versions and are not guaranteed to be backward compatible. However UDA version 2.0 is specified as a superset of UDA 1.1 and is thus backwards compatible with UDA 1.x versions. Therefore UDA 2.0 control points shall maintain interoperability with UDA 1.x devices. UDA 1.x control points can work with UDA 2.0 devices, but can't access the additional functionality specified in UDA 2.0. Version information is communicated in discovery and description messages. Discovery messages include the version of UPnP networking that the devices and control points support (in the SERVER and USER-AGENT header fields); the version of device and service types supported is also included in relevant discovery messages. Additionally, description documents also include the same information. SERVER and USER-AGENT header fields are also used in control and eventing to communicate which version of UPnP networking the devices and control points support. This clause explains the format of version information in discovery messages and specific requirements on discovery messages to maintain compatibility with advances in minor versions.

The remainder of this clause explains the UPnP discovery protocol known as SSDP (Simple Service Discovery Protocol) in detail, enumerating how devices advertise and revoke their advertisements as well as how control points search and devices respond.

## 1.1 SSDP message format

SSDP uses part of the header field format of HTTP 1.1 as defined in RFC 2616. However, it is NOT based on full HTTP 1.1 as it uses UDP instead of TCP, and it has its own processing rules. This subclause defines the generic format of a SSDP message.

All SSDP messages shall be formatted according to RFC 2616 clause 4.1 “generic message”. SSDP messages shall have a start-line and a list of message header fields. SSDP messages should not have a message body. If a SSDP message is received with a message body, the message body is allowed to be ignored.

### 1.1.1 SSDP Start-line

Each SSDP message shall have exactly one start-line. See clause 1.2, “Advertisement” and clause 1.3, “Search” below for the definition of all possible SSDP messages. The start-line shall be formatted either as defined in RFC 2616 clause 5.1 or clause 6.1. Furthermore, the start-line shall be one of the following three:

```
NOTIFY \* HTTP/1.1\r\nM-SEARCH \* HTTP/1.1\r\nHTTP/1.1 200 OK\r\n
```

As a clarification, while the start-line shall include “HTTP/1.1”, this does not signal that SSDP is fully based on HTTP 1.1; this start-line element is included for backward compatibility reasons only.

### 1.1.2 SSDP message header fields

The message header fields in a SSDP message shall be formatted according to RFC 2616 clause 4.2. This specifies that each message header field consist of a case-insensitive field name followed by a colon (":"), followed by the case-sensitive field value. SSDP restricts allowed field values.

Example SSDP header:

[HOST: 239.255.255.250:1900](http://HOST:239.255.255.250:1900)

### 1.1.3 SSDP header field extensions

UPnP working committees and UPnP vendors are allowed to extend SSDP messages with additional SSDP header fields. Additional message header fields can also be defined by the UPnP Forum Technical committee (e.g. clause 1.2, “Advertisement” defines BOOTID.UPNP.ORG, CONFIGID.UPNP.ORG, NEXTBOOTID.UPNP.ORG, and SEARCHPORT.UPNP.ORG header fields). To prevent name-clashes of header field definitions (two parties accidentally define the same header field name with different semantics), vendor-defined header field names shall have the following format:

field-name = token “.” domain-name

where the domain-name shall be Vendor Domain Name, and in addition shall satisfy the token format as defined in RFC 2616, clause 2.2.

Example vendor-defined SSDP header fields:

myheader.philips.com: “some value”  
myheader.sony.com: “other value”

### 1.1.4 UUID format and recommended generation algorithms

UPnP 2.0 devices shall format UUIDs according to the format specified below. However, UPnP 2.0 control points shall also be able to accept UUIDs that have not been formatted according to the rules specified below, as formatting rules are not specified in UPnP 1.0 other than the requirement that a UUID is a string.

UUIDs are 128 bit numbers that shall be formatted as specified by the following grammar (taken from [1]):

```
UUID = 4 * <hexOctet> “-” 2 * <hexOctet> “-” 2 * <hexOctet> “-” 2 * <hexOctet> “-” 6 * <hexOctet>
hexOctet = <hexDigit> <hexDigit>
hexDigit = “0”|“1”|“2”|“3”|“4”|“5”|“6”|“7”|“8”|“9”|“a”|“b”|“c”|“d”|“e”|“f”|“A”|“B”|“C”|“D”|“E”|“F”
```

The following is an example of the string representation of a UUID:

“2fac1234-31f8-11b4-a222-08002b34c003”

UUIDs are allowed to be generated using *any* suitable generation algorithm<sup>2</sup> that satisfies the following requirements:

- a) It is very unlikely to duplicate a UUID generated from some other resource.
- b) It maps down to a 128-bit number.
- c) UUIDs shall remain fixed over time.

The following UUID generation algorithm is recommended:

Time & MAC-based algorithm as specified in [1], where the UUID is generated once and stored in non-volatile memory if available.

### 1.1.5 SSDP processing rules

When an SSDP message is received that is not formatted according to clause 1.1, “SSDP message format” (the clauses above), receivers should silently discard the message. Receivers are allowed to try to parse such SSDP messages to try to interoperate.

---

<sup>2</sup> The UUID generation algorithm specified in [1] is RECOMMENDED, but is not MANDATORY, other UUID generation algorithms may be used instead, as long as they satisfy the three requirements.

When parsing SSDP header fields, receivers shall parse all required SSDP-defined header fields (see clause 1.2, “Advertisement” and clause 1.3, “Search” below) and are allowed to skip all other header fields. Receivers shall be able to skip header fields they do not understand.

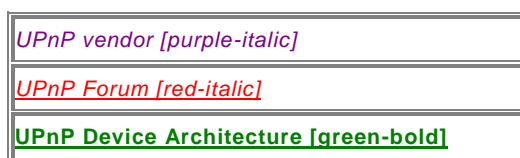
## 1.2 Advertisement

When a device is added to the network, the device advertises its services to control points. It does this by multicasting discovery messages to a standard address and port (239.255.255.250:1900). Control points listen to this port to detect when new capabilities are available on the network. To advertise the full extent of its capabilities, a device shall multicast a number of discovery messages corresponding to each of its root devices, embedded devices and services. Each message contains information specific to the embedded device (or service) as well as information about its enclosing device. Messages shall include duration until the advertisements expire; if the device remains available, the advertisements shall be re-sent (with new duration). If the device becomes unavailable, the device should explicitly cancel its advertisements, but if the device is unable to do this, the advertisements will expire on their own. If a multi-homed device becomes unavailable on some, but not all, of its UPnP-enabled interfaces, the device should explicitly cancel its advertisements on the affected UPnP-enabled interfaces (but NOT on the unaffected UPnP-enabled interfaces), but if the device is unable to do this, the advertisements on those interfaces or IP addresses will expire on their own. In addition, messages include the following header fields defined in this document: BOOTID.UPNP.ORG, NEXTBOOTID.UPNP.ORG, CONFIGID.UPNP.ORG, SEARCHPORT.UPNP.ORG. The field value of the BOOTID.UPNP.ORG header field shall be increased each time a device (re)joins the network and sends an initial announce (a “reboot” in UPnP terms), or adds a UPnP-enabled interface. Unless the device explicitly announces a change in the BOOTID.UPNP.ORG field value using an SSDP message, as long as the device remains continuously available in the network, the same BOOTID.UPNP.ORG field value shall be used in all repeat announcements, search responses, update messages and eventually bye-bye messages. Control points can parse this header field to detect whether the device has potentially lost its state (event subscriptions will have been lost, DCP specific state may have been changed) due to a “reboot”. Since a device cannot change IP addresses without changing the BOOTID.UPNP.ORG field value, the BOOTID.UPNP.ORG field value can also be used to distinguish multi-homed devices (in this case, a control point will see SSDP messages from different IP addresses with the same UUID, BOOTID.UPNP.ORG field value) from devices that changed IP addresses (in this case, the BOOTID.UPNP.ORG field value will be different). The field value of the NEXTBOOTID.UPNP.ORG header field indicates the field value of the BOOTID.UPNP.ORG header field that a multi-homed device intends to use in future announcements after adding a new UPnP-enabled interface. The field value of the CONFIGID.UPNP.ORG header field identifies the current set of device and service descriptions; control points can parse this header field to detect whether they need to send new *description* query messages. The field value of the SEARCHPORT.UPNP.ORG header field identifies the port at which the device listens to unicast M-SEARCH messages; control points can parse this header field to know to which port unicast M-SEARCH messages shall be sent. These header fields are explained in detail below.

### 1.2.1 Advertisement protocols and standards

To send (and receive) advertisements, devices (and control points) use the following subset of the overall UPnP protocol stack. (The overall UPnP protocol stack is listed at the beginning of this document.)

Figure 1-2: — Advertisement protocol stack





SSDP [blue]
UDP [black]
IP [black]

At the highest layer, discovery messages contain vendor-specific information, e.g., URL for the device description and device identifier. Moving down the stack, vendor content is supplemented by information from a UPnP Forum working committee, e.g., device type. Messages from the layers above are hosted in UPnP-specific protocols, defined in this document. In turn, the SSDP messages are delivered via UDP over IP. For reference, colors in [square brackets] above indicate which protocol defines specific header fields and field values in discovery messages listed below.

### 1.2.2 Device available - NOTIFY with `ssdp:alive`

When a device is added to the network, it shall multicast discovery messages to advertise its root device, any embedded devices, and any services. Each discovery message shall contain four major components:

- a) A notification type (e.g., device type), sent in an **NT** (Notification Type) header field.
- b) A composite identifier for the advertisement, sent in a **USN** (Unique Service Name) header field.
- c) A URL for more information about the device (or enclosing device in the case of a service), sent in a **LOCATION** header field.
- d) A duration for which the advertisement is valid, sent in a **CACHE-CONTROL** header field.

To advertise its capabilities, a device multicasts a number of discovery messages. Specifically, a root device shall multicast:

- Three discovery messages for the root device.

**Table 1-1 — Root device discovery messages**

	NT	USN <sup>a</sup>
1	<a href="#">upnp:rootdevice</a>	uuid: <i>device-UUID</i> :: <a href="#">upnp:rootdevice</a>
2	uuid: <i>device-UUID</i> <sup>b</sup>	uuid: <i>device-UUID</i> (for root device UUID)
3	urn: <a href="#">schemas-upnp-org:device:deviceType:ver</a> or urn: <i>domain-name</i> : <a href="#">device:deviceType:ver</a>	uuid: <i>device-UUID</i> ::urn: <a href="#">schemas-upnp-org:device:deviceType:ver</a> (of root device) or uuid: <i>device-UUID</i> ::urn: <i>domain-name</i> : <a href="#">device:deviceType:ver</a>
<sup>a</sup>	Note that the prefix of the <b>USN</b> header field (before the double colon) shall match the value of the <b>UDN</b> element in the device description. (Clause 2, "Description" explains the <b>UDN</b> element.)	
<sup>b</sup>	Note that the field value of this <b>NT</b> header field shall match the value of the <b>UDN</b> element in the device description.	

- Two discovery messages for each embedded device.

**Table 1-2 — Embedded device discovery messages**

	NT	USN <sup>a</sup>
1	uuid: <i>device-UUID</i> <sup>b</sup>	uuid: <i>device-UUID</i>
2	urn: <a href="#">schemas-upnp-org:device:d</a> <a href="#">eviceType:ver</a> or urn: <i>domain-name</i> : <a href="#">device:deviceType:ver</a>	uuid: <i>device-UUID</i> ::urn: <a href="#">schemas-upnp-org:device:deviceType:ver</a> or uuid: <i>device-UUID</i> ::urn: <i>domain-name</i> : <a href="#">device:deviceType:ver</a>

	NT	USN <sup>a</sup>
a	Note that the prefix of the <b>USN</b> header field (before the double colon) shall match the value of the <b>UDN</b> element in the device description. (Clause 2, “Description” explains the <b>UDN</b> element.)	
b	Note that the field value of this <b>NT</b> header field shall match the value of the <b>UDN</b> element in the device description	

- Once for each service type in each device.

**Table 1-3 — Service discovery messages**

	NT	USN <sup>a</sup>
1	urn: <u>schemas-upnp-</u> <u>org:service:serviceType:ver</u> or urn:domain-name: <u>service:serviceType:ver</u>	uuid:device-UUID::urn: <u>schemas-upnp-</u> <u>org:service:serviceType:ver</u> or uuid:device-UUID::urn:domain- name: <u>service:serviceType:ver</u>
a	Note that the field value of this <b>NT</b> header field shall match the value of the <b>UDN</b> element in the device description.	

If a root device has  $d$  embedded devices and  $s$  embedded services but only  $k$  distinct service types, this works out to  $3+2d+k$  requests. If a particular device or embedded device contains multiple instances of a particular service type, it is only necessary to advertise the service type once (rather than once for each instance). Note that if two embedded devices contain a service of the same service type, these services shall still be separately announced. This advertises the full extent of the device's capabilities to interested control points. These messages shall be sent out as a series with roughly comparable expiration times; order is unimportant, but refreshing or canceling individual messages is PROHIBITED.

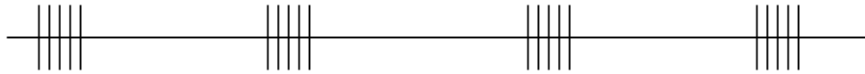
Updated UPnP device and service types are required to be fully backward compatible with previous versions of the same type. Devices shall advertise the highest supported version of each supported type. For example, if a device supports version 2 of the “Audio” service, it would advertise only version 2, even though it also supports version 1. It shall NOT advertise additional supported versions. Control points that support a given version of a device or service are able to also interact with higher versions because of this backward compatibility requirement, but only using the functionality that was defined in the lower version. For example, if a control point supports only version “1” of the “Audio” service, and a device advertises that it supports version “2” of the “Audio” service, the control point shall recognize the device and be able to use it.

Choosing an appropriate duration for advertisements is a balance between minimizing network traffic and maximizing freshness of device status. Relatively short durations close to the minimum of 1800 seconds will ensure that control points have current device status at the expense of additional network traffic; longer durations, say on the order of a day, compromise freshness of device status but can significantly reduce network traffic. Generally, device vendors should choose a value that corresponds to expected device usage: short durations for devices that are expected to be part of the network for short periods of time, and significantly longer durations for devices expected to be long-term members of the network. Devices that frequently connect to and leave the network (such as mobile wireless devices) should use a shorter duration so that control points have a more accurate view of their availability. Advertisements in a set (both initial and subsequent) should have comparable durations. Advertisements in the initial set should be sent as quickly as possible. Subsequent refreshments of the advertisements are allowed to be spread over time rather than being sent as a group.

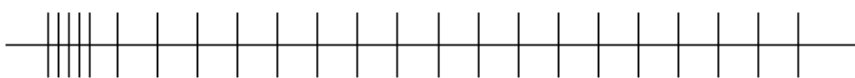
Spreading refreshments of advertisements over time rather than being sent as a group improves reliability in case there are network glitches: without increasing the total network load it increases the frequency of sending announcements from devices to control points. The two figures below show the announcement behavior without spreading and with spreading the

messages over the entire interval. The figures show a timeline from the moment a device joins the network, sends its initial announcements (represented by vertical lines), and subsequently periodically sends repeat announcements. In the second figure, these repeat announcements are spread over the entire period rather than sent as a bunch.

**Figure 1-3: — Initial and repeat announcements, no announcement spreading**



**Figure 1-4: — Initial and repeat announcements, message spreading of repeat announcements**



Devices should wait a random interval (e.g. between 0 and 100milliseconds) before sending an initial set of advertisements in order to reduce the likelihood of network storms; this random interval should also be applied on occasions where the device obtains a new IP address or a new UPnP-enabled interface is installed.

Due to the unreliable nature of UDP, devices should send the entire set of discovery messages more than once with some delay between sets e.g. a few hundred milliseconds. To avoid network congestion discovery messages should not be sent more than three times. In addition, the device shall re-send its advertisements periodically prior to expiration of the duration specified in the **CACHE-CONTROL** header field; it is Recommended that such refreshing of advertisements be done at a randomly-distributed interval of less than one-half of the advertisement expiration time, so as to provide the opportunity for recovery from lost advertisements before the advertisement expires, and to distribute over time the advertisement refreshment of multiple devices on the network in order to avoid spikes in network traffic. Note that UDP packets are also bounded in length (perhaps as small as 512 Bytes in some implementations); each discovery message shall fit entirely in a single UDP packet. There is no guarantee that the above  $3+2d+k$  messages will arrive in a particular order.

A multi-homed device shall perform the above announcement procedures on each of its UPnP-enabled interfaces. Advertisements sent on multiple UPnP-enabled interfaces shall contain the same field values except for the HOST, CACHE-CONTROL and LOCATION header fields. The HOST field value of an advertisement shall be the standard multicast address specified for the protocol (IPv4 or IPv6) used on the interface. The URL specified by the LOCATION header field shall be reachable on the interface on which the advertisement is sent. Finally, advertisements sent on different interfaces are allowed to have different CACHE-CONTROL field values and are allowed to be sent with different frequencies.

When a device is added to the network, it shall send a multicast message with method **NOTIFY** and **ssdp:alive** in the **NTS** header field in the following format. Values in *italics* are placeholders for actual values.

```
NOTIFY * HTTP/1.1
HOST: 239.255.255.250:1900
CACHE-CONTROL: max-age = seconds until advertisement expires
LOCATION: URL for UPnP description for root device
NT: notification type
NTS: ssdp:alive
SERVER: OS/version UPnP/2.0 product/version
USN: composite identifier for the advertisement
BOOTID.UPNP.ORG: number increased each time device sends an initial announce or an update message
CONFIGID.UPNP.ORG: number used for caching description information
SEARCHPORT.UPNP.ORG: number identifies port on which device responds to unicast M-SEARCH
```

Note: No body is sent for messages with method **NOTIFY**, but note that the message shall have a blank line following the last header field.

The TTL for the IP packet should default to 2 and should be configurable.

Listed below are details for the request line and header fields appearing in the listing above. Field names are not case sensitive. All field values are case sensitive except where noted.

## Request line

Shall be "NOTIFY \* HTTP/1.1"

### NOTIFY

Method for sending notifications and events.

\*

Message applies generally and not to a specific resource. shall be \*.

### HTTP/1.1

HTTP version.

## Header fields

### HOST

Required. Field value contains multicast address and port reserved for SSDP by Internet Assigned Numbers Authority (IANA). shall be [239.255.255.250:1900](#). If the port number (":1900") is omitted, the receiver shall assume the default SSDP port number of 1900.

### CACHE-CONTROL

Required. Field value shall have the max-age directive ("max-age=") followed by an integer that specifies the number of seconds the advertisement is valid. After this duration, control points should assume the device (or service) is no longer available; as long as a control point has received at least one advertisement that is still valid from a root device, any of its embedded devices or any of its services, then the control point can assume that all are available. The number of seconds should be greater than or equal to 1800 seconds (30 minutes), although exceptions are defined in the text above. Specified by UPnP vendor. Other directives shall NOT be sent and shall be ignored when received.

### LOCATION

Required. Field value contains a URL to the UPnP description of the root device. Normally the host portion contains a literal IP address rather than a domain name in unmanaged networks. Specified by UPnP vendor. Single absolute URL (see RFC 3986).

### NT

Required. Field value contains Notification Type. shall be one of the following. (See Table 1-1, "Root device discovery messages", Table 1-2, "Embedded device discovery messages", and Table 1-3, "Service discovery messages" above.) Single URI.

#### [upnp:rootdevice](#)

Sent once for root device.

#### [uuid:device-UUID](#)

Sent once for each device, root or embedded, where *device-UUID* is specified by the UPnP vendor. See clause 1.1.4, "UUID format and recommended generation algorithms" for the MANDATORY UUID format.

#### [urn:schemas-upnp-org:device:deviceType:ver](#)

Sent once for each device, root or embedded, where *deviceType* and *ver* are defined by UPnP Forum working committee, and *ver* specifies the version of the device type.

#### [urn:schemas-upnp-org:service:serviceType:ver](#)

Sent once for each service where *serviceType* and *ver* are defined by UPnP Forum working committee and *ver* specifies the version of the service type.

#### [urn:domain-name:device:deviceType:ver](#)

Sent once for each device, root or embedded, where *domain-name* is a Vendor Domain Name, *deviceType* and *ver* are defined by the UPnP vendor, and *ver* specifies the version of the device type. Period characters in the Vendor Domain Name shall be replaced with hyphens in accordance with RFC 2141.

urn:*domain-name*:*service*:*serviceType*:*ver*

Sent once for each service where *domain-name* is a Vendor Domain Name, *serviceType* and *ver* are defined by UPnP vendor, and *ver* specifies the version of the service type. Period characters in the Vendor Domain Name shall be replaced with hyphens in accordance with RFC 2141.

#### NTS

Required. Field value contains Notification Sub Type. shall be [ssdp:alive](#). Single URI.

#### SERVER

Required. Specified by UPnP vendor. String. Field value shall begin with the following "product tokens" (defined by HTTP/1.1). The first product token identifies the operating system in the form *OS name/OS version*, the second token represents the UPnP version and shall be [UPnP/2.0](#), and the third token identifies the product using the form *product name/product version*. For example, "SERVER: *unix/5.1* [UPnP/2.0](#) *MyProduct/1.0*".

#### USN

Required. Field value contains Unique Service Name. Identifies a unique instance of a device or service. shall be one of the following. (See Table 1-1, "Root device discovery messages", Table 1-2, "Embedded device discovery messages", and Table 1-3, "Service discovery messages" above.) The prefix (before the double colon) shall match the value of the UDN element in the device description. (Clause 2, "Description" explains the UDN element.) Single URI.

uuid:*device-UUID*::[upnp:rootdevice](#)

Sent once for root device where *device-UUID* is specified by UPnP vendor. See clause 1.1.4, "UUID format and recommended generation algorithms" for the MANDATORY UUID format.

uuid:*device-UUID*

Sent once for every device, root or embedded, where *device-UUID* is specified by the UPnP vendor. See clause 1.1.4, "UUID format and recommended generation algorithms" for the MANDATORY UUID format.

uuid:*device-UUID*::urn:[schemas-upnp-org:device](#):*deviceType*:*ver*

Sent once for every device, root or embedded, where *device-UUID* is specified by the UPnP vendor, *deviceType* and *ver* are defined by UPnP Forum working committee and *ver* specifies version of the device type. See clause 1.1.4, "UUID format and recommended generation algorithms" for the MANDATORY UUID format.

uuid:*device-UUID*::urn:[schemas-upnp-org:service](#):*serviceType*:*ver*

Sent once for every service where *device-UUID* is specified by the UPnP vendor, *serviceType* and *ver* are defined by UPnP Forum working committee and *ver* specifies version of the device type. See clause 1.1.4, "UUID format and recommended generation algorithms" for the MANDATORY UUID format.

uuid:*device-UUID*::urn:*domain-name*:[device](#):*deviceType*:*ver*

Sent once for every device, root or embedded, where *device-UUID*, *domain-name* (a Vendor Domain Name), *deviceType* and *ver* are defined by the UPnP vendor and *ver* specifies the version of the device type. See clause 1.1.4, "UUID format and recommended generation algorithms" for the MANDATORY UUID format. Period characters in the Vendor Domain Name shall be replaced by hyphens in accordance with RFC 2141.

uuid:*device-UUID*::urn:*domain-name*:[service](#):*serviceType*:*ver*

Sent once for every service where *device-UUID*, *domain-name* (a Vendor Domain Name), *serviceType* and *ver* are defined by the UPnP vendor and *ver* specifies the version of the service type. See clause 1.1.4, "UUID format and recommended generation algorithms" for the MANDATORY UUID format. Period characters in the Vendor Domain Name shall be replaced by hyphens in accordance with RFC 2141.

#### BOOTID.UPNP.ORG

Required. The BOOTID.UPNP.ORG header field represents the boot instance of the device expressed according to a monotonically increasing value. Its field value shall be a non-negative 31-bit integer; ASCII encoded, decimal, without leading zeros (leading zeroes, if present, shall be ignored by the recipient) that shall be increased on each initial announce of the UPnP device or shall be the same as the field value of the NEXTBOOTID.UPNP.ORG header field in the last sent SSDP update message. Its field value shall remain the same on all periodically repeated announcements. A convenient mechanism is to set this field value to the time that the device sends its initial announcement, expressed as seconds elapsed since midnight January 1, 1970; for devices that have a notion of time, this will not require any additional state to remember or be "flushed". However, it is perfectly acceptable to use a simple boot counter that is incremented on every initial

announcement as a field value of this header field. As such, control points shall NOT view this header field as a timestamp. The BOOTID.UPNP.ORG header field shall be included in *all* announcements of a root device, its embedded devices and its services. Unless the device explicitly updates its value by sending an SSDP update message, as long as the device remains available in the network, the same BOOTID.UPNP.ORG field value shall be used in *all* announcements, search responses, update messages and eventually bye-bye messages.

Control points can use this header field to detect the case when a device leaves and rejoins the network (“reboots” in UPnP terms). It can be used by control points for a number of purposes such as re-establishing desired event subscriptions, checking for changes to the device state that were not evented since the device was off-line.

#### CONFIGID.UPNP.ORG

Required. The CONFIGID.UPNP.ORG field value shall be a non-negative, 31-bit integer, ASCII encoded, decimal, without leading zeros (leading zeroes, if present, shall be ignored by the recipient) that shall represent the configuration number of a root device. UPnP 2.0 devices are allowed to be freely assign configid numbers from 0 to 16777215 ( $2^{24}-1$ ). Higher numbers are reserved for future use, and can be assigned by the Technical Committee. The **configuration** of a root device consists of the following information: the DDD of the root device and all its embedded devices, and the SCPDs of all the contained services. If any part of the configuration changes, the CONFIGID.UPNP.ORG field value shall be changed. The CONFIGID.UPNP.ORG header field shall be included in all announcements of a root device, its embedded devices and its services. The configuration number that is present in a CONFIGID.UPNP.ORG field value shall satisfy the following rule:

- if a device sends out two messages with a CONFIGID.UPNP.ORG header field with the same field value K, the configuration shall be the same at the moments that these messages were sent.

Whenever a control point receives a CONFIGID.UPNP.ORG header field with a field value K, and subsequently downloads the configuration information, this configuration information is associated with K. As an additional safeguard, the device shall include a `configId` attribute with value K in the returned description (see clause 2, “Description”). The following caching rules for control points supersede the caching rules that are defined in UPnP 1.0:

- Control points are allowed to ignore the CONFIGID.UPNP.ORG header field and use the caching rules that are based on advertisement expirations as defined in Clause 2, Description: as long as at least one of the discovery advertisements from a root device, its embedded devices and its services have not expired, a control point is allowed to assume that the root device and all its embedded devices and all its services are available. The device and service descriptions are allowed to be retrieved at any point since the device and service descriptions are static as long as the device and its services are available.
- If no configuration number is included in a received SSDP message, control points should cache based on advertisement expirations as defined in Clause 2 Description.
- If a CONFIGID.UPNP.ORG header field with field value K is included in a received SSDP message, and a control point has already cached information associated with field value K, the control point is allowed to use this cached information as the current configuration of the device. Otherwise, a control point should assume it has not cached the current configuration of the device and needs to send new description query messages.

The CONFIGID.UPNP.ORG header field reduces peak loads on UPnP devices during startup and during network hiccups. Only if a control point receives an announcement of an unknown configuration is downloading required.

#### SEARCHPORT.UPNP.ORG

Allowed. If a device does not send the SEARCHPORT.UPNP.ORG header field, it shall respond to unicast M-SEARCH messages on port 1900. Only if port 1900 is unavailable it is allowed for a device select a different port to respond to unicast M-SEARCH messages. If a device sends the SEARCHPORT.UPNP.ORG header field, its field value shall be an ASCII encoded integer, decimal, without leading zeros (leading zeroes, if present, shall be ignored by the recipient), in the range 49152-65535 (RFC 4340). The device shall respond to unicast M-SEARCH messages that are sent to the advertised port.

#### SECURELOCATION.UPNP.ORG

Allowed. Required when Device Protection is implemented.

The SECURELOCATION.UPNP.ORG header shall provide a base URL with “https:” for the scheme component and indicate the correct “port” subcomponent in the “authority” component for a TLS connection. Because the scheme and authority components are not included in relative URLs, these components are obtained from the base URL provided by either LOCATION or SECURELOCATION.UPNP.ORG. See for more information Ref DEVICEPROTECTION.

Note: No responses are sent for messages with method NOTIFY.

### 1.2.3 Device unavailable -- NOTIFY with ssdp:byebye

When a device and its services are going to be removed from the network, the device should multicast an **ssdp:byebye** message corresponding to *each* of the **ssdp:alive** messages it multicast that have not already expired. If the device is removed abruptly from the network, it might not be possible to multicast a message. As a fallback, discovery messages shall include an expiration value in a CACHE-CONTROL field value (as explained above); if not re-advertised, the discovery message eventually expires on its own.

(Note: when a control point is about to be removed from the network, no discovery-related action is required.)

When a device is about to be removed from the network, it should explicitly revoke its discovery messages by sending one multicast message for *each* **ssdp:alive** message it sent. Each multicast message shall have method **NOTIFY** and **ssdp:byebye** in the **NTS** header field in the following format. Values in *italics* are placeholders for actual values.

When a multi-homed device is about to be removed from the network on one or more of its UPnP-enabled interfaces, it should explicitly revoke its discovery messages by sending one multicast message for *each* **ssdp:alive** message it has previously sent on those interfaces and IP addresses. It shall NOT send such multicast messages to any of the UPnP-enabled interfaces that remain available.

When **ssdp:byebye** messages are sent on multiple UPnP-enabled interfaces, the messages shall contain identical field values except for the HOST field value. The HOST field value of an advertisement shall be the standard multicast address specified for the protocol (IPv4 or IPv6) used on the interface.

```
NOTIFY * HTTP/1.1
HOST: 239.255.255.250:1900
NT: notification type
NTS: ssdp:byebye
USN: composite identifier for the advertisement
BOOTID.UPNP.ORG: number increased each time device sends an initial announce or an update message
CONFIGID.UPNP.ORG: number used for caching description information
```

Note: No body is present for messages with method NOTIFY, but note that the message shall have a blank line following the last header field.

The TTL for the IP packet should default to 2 and should be configurable.

Listed below are details for the request line and header fields appearing in the listing above. Field names are not case sensitive. All field values are case sensitive except where noted.

#### Request line

Shall be "NOTIFY \* HTTP/1.1"

NOTIFY

Method for sending notifications and events.

\*

Message applies generally and not to a specific resource. shall be \*.

HTTP/1.1

HTTP version.

#### Header fields



#### HOST

Required. Field value contains multicast address and port reserved for SSDP by Internet Assigned Numbers Authority (IANA). shall be [239.255.255.250:1900](#). If the port number (“:1900”) is omitted, the receiver shall assume the default SSDP port number of 1900.

#### NT

Required. Field value contains Notification Type. (See list of required field values for the NT header field in NOTIFY with `ssdp:alive` above.) Single URI.

#### NTS

Required. Field value contains Notification Sub Type. shall be [ssdp:byebye](#). Single URI.

#### USN

Required. Field value contains Unique Service Name. (See list of required field values for the USN header field in NOTIFY with `ssdp:alive` above.) Single URI.

#### BOOTID.UPNP.ORG

Required. As defined in clause 1.2, and 1.2.2.

#### CONFIGID.UPNP.ORG

Required. As defined in clause 1.2, and 1.2.2.

Note: No responses are sent for messages with method NOTIFY.

If a control point has received *at least one* `ssdp:byebye` message of a root device, any of its embedded devices or any of its services then the control point can assume that all are no longer available. As a fallback, if a control point fails to receive notification that a root device, its embedded devices and its services are unavailable, the original discovery messages will eventually expire yielding the same effect. Only when *all* original advertisements of a root device, its embedded devices and its services have expired can a control point assume that they are no longer available.

If a multi-homed control point has received *at least one* `ssdp:byebye` message of a root device, any of its embedded devices or any of its services on one of its UPnP-enabled interfaces then the control point can assume that all are no longer available on that UPnP-enabled interface. However, the control point shall NOT assume that the device is also no longer available on all of its other UPnP-enabled interfaces. As a fallback, if a control point fails to receive notification that a root device, its embedded devices and its services are unavailable on a particular UPnP-enabled interface, the original discovery messages will eventually expire yielding the same effect. Only when *all* original advertisements of a root device, its embedded devices and its services received on a UPnP-enabled interface have expired can a control point assume that they are no longer available on that interface or IP address.

#### 1.2.4 Device Update – NOTIFY with `ssdp:update`

When a new UPnP-enabled interface is added to a multi-homed device, the device shall increase its BOOTID.UPNP.ORG field value, multicast an **`ssdp:update`** message for each of the root devices, embedded devices and embedded services to all of the existing UPnP-enabled interfaces to announce a change in the BOOTID.UPNP.ORG field value, and re-advertise itself on all (existing and new) UPnP-enabled interfaces with the new BOOTID.UPNP.ORG field value. Similarly, if a multi-homed device loses connectivity on a UPnP-enabled interface and regains connectivity, or if the IP address on one of the UPnP-enabled interfaces changes, the device shall increase the BOOTID.UPNP.ORG field value, multicast an **`ssdp:update`** message for each of the root devices, embedded devices and embedded services to all the unaffected UPnP-enabled interfaces to announce a change in the BOOTID.UPNP.ORG field value, and re-advertise itself on all (affected and unaffected) UPnP-enabled interfaces with the new BOOTID.UPNP.ORG field value. In all cases, the **`ssdp:update`** message for the root devices shall be sent as soon as possible. Other **`ssdp:update`** messages should be spread over time. However, all **`ssdp:update`** messages shall be sent before any announcement messages with the new BOOTID.UPNP.ORG field value can be sent.

When **ssdp:update** messages are sent on multiple UPnP-enabled interfaces, the messages shall contain identical field values except for the HOST and LOCATION field values. The HOST field value of an advertisement shall be the standard multicast address specified for the protocol (IPv4 or IPv6) used on the interface. The URL specified in the LOCATION field value shall be reachable on the interface on which the advertisement is sent.

```
NOTIFY * HTTP/1.1
HOST: 239.255.255.250:1900
LOCATION: URL for UPnP description for root device
NT: notification type
NTS: ssdp:update
USN: composite identifier for the advertisement
BOOTID.UPNP.ORG: BOOTID value that the device has used in its previous announcements
CONFIGID.UPNP.ORG: number used for caching description information
NEXTBOOTID.UPNP.ORG: new BOOTID value that the device will use in subsequent announcements
SEARCHPORT.UPNP.ORG: number identifies port on which device responds to unicast M-SEARCH
```

Note: No body is present for messages with method NOTIFY, but note that the message shall have a blank line following the last header field.

The TTL for the IP packet should default to 2 and should be configurable.

Listed below are details for the request line and header fields appearing in the listing above. Field names are not case sensitive. All field values are case sensitive except where noted.

### Request line

Shall be "NOTIFY \* HTTP/1.1"

#### NOTIFY

Method for sending notifications and events.

\*

Message applies generally and not to a specific resource. Shall be \*.

#### HTTP/1.1

HTTP version.

### Header fields

#### HOST

Required. Field value contains multicast address and port reserved for SSDP by Internet Assigned Numbers Authority (IANA). Shall be [239.255.255.250:1900](#). If the port number (":1900") is omitted, the receiver shall assume the default SSDP port number of 1900.

#### LOCATION

Required. Field value shall be the same as the LOCATION field value that has been sent in previous SSDP messages. Single absolute URL (see RFC 3986).

#### NT

Required. Field value contains Notification Type. (See list of required field values for the NT header field in NOTIFY with `ssdp:alive` above.) Single URI.

#### NTS

Required. Field value contains Notification Sub Type. Shall be [ssdp:update](#). Single URI.

#### USN

Required. Field value contains Unique Service Name. (See list of required field values for the USN header field in NOTIFY with `ssdp:alive` above.) Single URI.

#### BOOTID.UPNP.ORG

Required. As defined in clause 1.2, and 1.2.2, Field value shall be the same as the BOOTID.UPNP.ORG field value that has been sent in previous SSDP messages.

#### CONFIGID.UPNP.ORG

Required. As defined in clause 1.2, and 1.2.2.

#### NEXTBOOTID.UPNP.ORG

Required. Field value contains the new BOOTID.UPNP.ORG field value that the device intends to use in the subsequent device and service announcement messages. Its field value shall be a non-negative 31-bit integer; ASCII encoded, decimal, without leading zeros (leading zeroes, if present, shall be ignored by the recipient) and shall be greater than the field value of the BOOTID.UPNP.ORG header field.

#### SEARCHPORT.UPNP.ORG

Allowed. As defined in clause 1.2, and 1.2.2.

#### SECURELOCATION.UPNP.ORG

Allowed. As defined in section 1.2.2.

Note: No responses are sent for messages with method NOTIFY.

If a control point with a single UPnP-enabled interface receives an **ssdp:update** message, the NEXTBOOTID.UPNP.ORG field value replaces the BOOTID.UPNP.ORG field value that the control point has previously recorded for the device. It can expect future announcements, search responses, update messages and eventually bye-bye messages from the device to contain the “new” BOOTID.UPNP.ORG field value (that is: the field value of the NEXTBOOTID.UPNP.ORG header field in the received **ssdp:update** message). The field value in the NEXTBOOTID.UPNP.ORG header field shall be recorded as the current BOOTID.UPNP.ORG field value of the device which is to be expected on all subsequent SSDP messages.

If a multi-homed control point receives an **ssdp:update** message on its UPnP-enabled interface(s), and the message arrives on the interface(s) that it uses for UPnP communications with the device (such as event subscriptions), it can assume that the device has remained continuously available (including all device state), and that the NEXTBOOTID.UPNP.ORG field value replaces the BOOTID.UPNP.ORG field value that the control point has previously recorded for the device. It can expect future announcements, search responses, update messages and eventually bye-bye messages from the device to contain the “new” BOOTID.UPNP.ORG field value (that is: the field value of the NEXTBOOTID.UPNP.ORG header field in the received **ssdp:update** message). The field value in the NEXTBOOTID.UPNP.ORG header field shall be recorded as the current BOOTID.UPNP.ORG field value of the device which is to be expected on all subsequent SSDP messages.

If a control point receives an SSDP message with a BOOTID.UPNP.ORG field value different (either higher or lower) from the value that the control point has previously recorded for the device, it can assume that the device has become temporarily unavailable on that interface and has become available again, and any stored state information about the device has become invalid. It shall treat the device as a newly discovered device.

### 1.3 Search

When a control point is added to the network, the UPnP discovery protocol allows that control point to search for devices of interest on the network. It does this by multicasting on the reserved address and port (239.255.255.250:1900) a search message with a pattern, or target, equal to a type or identifier for a device or service. Responses from devices contain discovery messages essentially identical to those advertised by newly connected devices; the former are unicast while the latter are multicast. Control points can also send a unicast search message to a known IP address and port 1900 or the port indicated by SEARCHPORT.UPNP.ORG, to verify the existence of UPnP device(s) and service(s) at the IP address. For example, a unicast search may be used to quickly check whether a known UPnP device or service is still available on the network. Multi-homed control points are allowed to choose to send discovery messages on any, some or all of its UPnP-enabled interfaces.

### 1.3.1 Search protocols and standards

To search for devices (and be discovered by control points), control points (and devices) use the following subset of the overall UPnP protocol stack. (The overall UPnP protocol stack is listed at the beginning of this document.)

**Figure 1-5: — Search protocol stack**

<i>UPnP vendor</i> [purple-italic]
<i>UPnP Forum</i> [red-italic]
<b>UPnP Device Architecture</b> [green-bold]
SSDP [blue]
UDP [black]
IP [black]

At the highest layer, search messages contain vendor-specific information, e.g., the control point, device, and service identifiers. Moving down the stack, vendor content is supplemented by information from a UPnP Forum working committee, e.g., device or service types. Messages from the layers above are hosted in UPnP-specific protocols, defined in this document. In turn, search requests are delivered via multicast and unicast SSDP messages defined in this document. Search responses are delivered via a unicast SSDP messages defined in this document. Both kinds of messages are delivered via UDP over IP. For reference, colors in [square brackets] above indicate which protocol defines specific header fields and field values in discovery messages listed below.

### 1.3.2 Search request with M-SEARCH

When a control point desires to search the network for devices, it shall send a multicast request with method **M-SEARCH** in the following format. Control points that know the address of a specific device are allowed to also use a similar format to send unicast requests with method M-SEARCH.

For multicast M-SEARCH, the message format is defined below. Values in *italics* are placeholders for actual values.

```
M-SEARCH * HTTP/1.1
HOST: 239.255.255.250:1900
MAN: "ssdp:discover"
MX: seconds to delay response
ST: search target
USER-AGENT: OS/version UPnP/2.0 product/version
CPFN.UPNP.ORG: friendly name of the control point
CPUUID.UPNP.ORG: uuid of the control point
```

Note: No body is present in requests with method **M-SEARCH**, but note that the message shall have a blank line following the last header field.

Note: The TTL for the IP packet should default to 2 and should be configurable.

Listed below are details for the request line and header fields appearing in the listing above. Field names are not case sensitive. All field values are case sensitive except where noted.

#### Request line

Shall be "M-SEARCH \* HTTP/1.1"

M-SEARCH

Method for search requests.

\*

Request applies generally and not to a specific resource. shall be \*.

HTTP/1.1

HTTP version.

## Header fields

HOST

Required. Field value contains the multicast address and port reserved for SSDP by Internet Assigned Numbers Authority (IANA). shall be [239.255.255.250:1900](#).

MAN

Required by HTTP Extension Framework. Unlike the NTS and ST field values, the field value of the MAN header field is enclosed in double quotes; it defines the scope (namespace) of the extension. shall be "[ssdp:discover](#)".

MX

Required. Field value contains maximum wait time in seconds. shall be greater than or equal to 1 and should be less than 5 inclusive. Device responses should be delayed a random duration between 0 and this many seconds to balance load for the control point when it processes responses. This value is allowed to be increased if a large number of devices are expected to respond. The MX field value should NOT be increased to accommodate network characteristics such as latency or propagation delay (for more details, see the explanation below). Specified by UPnP vendor. Integer.

ST

Required. Field value contains Search Target. shall be one of the following. (See NT header field in NOTIFY with [ssdp:alive](#) above.) Single URI.

[ssdp:all](#)

Search for all devices and services.

[upnp:rootdevice](#)

Search for root devices only.

uuid:*device-UUID*

Search for a particular device. *device-UUID* specified by UPnP vendor. See clause 1.1.4, "UUID format and recommended generation algorithms" for the MANDATORY UUID format.

urn:[schemas-upnp-org:device:deviceType:ver](#)

Search for any device of this type where [deviceType](#) and [ver](#) are defined by the UPnP Forum working committee.

urn:[schemas-upnp-org:service:serviceType:ver](#)

Search for any service of this type where [serviceType](#) and [ver](#) are defined by the UPnP Forum working committee.

urn:[domain-name:device:deviceType:ver](#)

Search for any device of this typewhere *domain-name* (a Vendor Domain Name), *deviceType* and *ver* are defined by the UPnP vendor and *ver* specifies the highest supported version of the device type. Period characters in the Vendor Domain Name shall be replaced with hyphens in accordance with RFC 2141.

urn:[domain-name:service:serviceType:ver](#)

Search for any service of this type. Where *domain-name* (a Vendor Domain Name), *serviceType* and *ver* are defined by the UPnP vendor and *ver* specifies the highest supported version of the service type. Period characters in the Vendor Domain Name shall be replaced with hyphens in accordance with RFC 2141.

USER-AGENT

Allowed. Specified by UPnP vendor. String. Field value shall begin with the following "product tokens" (defined by HTTP/1.1). The first product token identifies the operating system in the form *OS name/OS version*, the second token represents the UPnP version and shall be [UPnP/2.0](#), and the third token identifies the product using the form *product name/product version*. For example, "USER-AGENT: *unix/5.1* [UPnP/2.0](#) *MyProduct/1.0*".

#### TCPPORT.UPNP.ORG

Allowed. A control point can request that a device replies to a TCP port on the control point. When this header is present it identifies the TCP port on which the device can reply to the search. If a control point sends the TCPPORT.UPNP.ORG header field, its field value shall be an ASCII encoded integer, decimal, without leading zeros (leading zeroes, if present, shall be ignored by the recipient), in the range 49152-65535 (RFC 4340). The device shall respond to unicast M-SEARCH messages similar to sending the response to the originating UDP port except that the notification messages are sent to the advertised TCPPORT.UPNP.ORG port over TCP instead of UDP.

#### CPFN.UPNP.ORG

Required. Specifies the friendly name of the control point. The friendly name is vendor specific. When Device Protection is implemented the cpfn.upnp.org shall be the same as the <Name> of Device Protection unless the Device Protection <Alias> is defined, in which case it shall use the <Alias>.

#### CPUUID.UPNP.ORG

Allowed. uuid of the control point. When the control point is implemented in a UPnP device it is recommended to use the UDN of the co-located UPnP device. When implemented, all specified requirements for uuid usage in devices also apply for control points. See section 1.1.4. Note that when Device Protection is implemented the CPUUID.UPNP.ORG shall be the same as the uuid used in Device Protection.

For unicast M-SEARCH, the message format is defined below. Values in *italics* are placeholders for actual values.

```
M-SEARCH * HTTP/1.1
HOST: hostname:portNumber
MAN: "ssdp:discover"
ST: search target
USER-AGENT: OS/version UPnP/2.0 product/version
```

Note: No body is present in requests with method **M-SEARCH**, but note that the message shall have a blank line following the last header field.

Listed below are details for the request line and header fields appearing in the listing above. Field names are not case sensitive. All field values are case sensitive except where noted.

#### Request line

Shall be "[M-SEARCH \\* HTTP/1.1](#)"

#### M-SEARCH

Method for search requests.

\*

Request applies generally and not to a specific resource. Shall be \*.

#### HTTP/1.1

HTTP version.

#### Header fields

##### HOST

Required. For unicast requests, the field value shall be the domain name or IP address of the target device and either port 1900 or the SEARCHPORT provided by the target device.

##### MAN

Required by HTTP Extension Framework. Unlike the NTS and ST field values, the field value of the MAN header field is enclosed in double quotes; it defines the scope (namespace) of the extension. Shall be "[ssdp:discover](#)".

##### ST

Required. Field value contains Search Target. Shall be one of the following. (See NT header field in NOTIFY with [ssdp:alive](#) above.) Single URI.

[ssdp:all](#)

Search for all devices and services.

[upnp:rootdevice](#)

Search for root devices only.

[uuid:device-UUID](#)

Search for a particular device. *device-UUID* specified by UPnP vendor. See clause 1.1.4, “UUID format and recommended generation algorithms” for the MANDATORY UUID format.

[urn:schemas-upnp-org:device:deviceType:ver](#)

Search for any device of this type where *deviceType* and *ver* are defined by the UPnP Forum working committee.

[urn:schemas-upnp-org:service:serviceType:ver](#)

Search for any service of this type where *serviceType* and *ver* are defined by the UPnP Forum working committee.

[urn:domain-name:device:deviceType:ver](#)

Search for any device of this type where *domain-name* (a Vendor Domain Name), *deviceType* and *ver* are defined by the UPnP vendor and *ver* specifies the highest supported version of the device type. Period characters in the Vendor Domain Name shall be replaced with hyphens in accordance with RFC 2141.

[urn:domain-name:service:serviceType:ver](#)

Search for any service of this type where *domain-name* (a Vendor Domain Name), *serviceType* and *ver* are defined by the UPnP vendor and *ver* specifies the highest supported version of the service type. Period characters in the Vendor Domain Name shall be replaced with hyphens in accordance with RFC 2141.

#### USER-AGENT

Allowed. Specified by UPnP vendor. String. Field value shall begin with the following “product tokens” (defined by HTTP/1.1). The first product token identifies the operating system in the form *OS name/OS version*, the second token represents the UPnP version and shall be [UPnP/2.0](#), and the third token identifies the product using the form *product name/product version*. For example, “USER-AGENT: *unix/5.1* [UPnP/2.0](#) *MyProduct/1.0*”.

Due to the unreliable nature of UDP, control points should send each M-SEARCH message more than once. As a fallback, to guard against the possibility that a device might not receive the M-SEARCH message from a control point, a device should re-send its advertisements periodically (see **CACHE-CONTROL** header field in **NOTIFY** with `ssdp:alive` above).

For a multicast request, the control point should wait at least the amount of time specified in the MX header field for responses to arrive from devices. The random distribution of responses over the MX interval means that a responder is allowed to send a response at MX seconds after receiving the M-SEARCH request. The MX field value is allowed to be adjusted by heuristics at the requester based on, for example, observed number of responders. Network characteristics affecting the propagation of traffic cannot be addressed by increasing the MX field value because of the reason cited above. A requester is allowed to adapt to network characteristics with heuristics based on observed network behavior (the exact heuristics are out of scope). The net effect is that the M-SEARCH request persists at the requester for a period of time exceeding MX such that the characteristics of the network are properly accommodated to minimize lost responses.

When a device receives a unicast M-SEARCH, it should respond within 1 second and it is allowed to respond sooner. The sender of the unicast request should wait at least 1 second for the response.

Updated versions of device and service types are required to be fully backward compatible with previous versions. Devices shall respond to M-SEARCH requests for any supported version. For example, if a device implements “[urn:schemas-upnp-org:service:xyz:2](#)”, it shall respond to search requests for both that type and “[urn:schemas-upnp-org:service:xyz:1](#)”. The response shall specify the same version as was contained in the search request. If a control

point searches for a device or service of a particular version and receives no responses (presumably because no device present on the network supports the specified version), but is willing to operate using a lower version, it is allowed to repeat the search request specifying the lower version.

### 1.3.3 Search response

To be found by a network search, a device shall send a unicast UDP response to the source IP address and port that sent the request to the multicast address. Devices respond if the ST header field of the M-SEARCH request is “ssdp:all”, “upnp:rootdevice”, “uuid:” followed by a UUID that exactly matches the one advertised by the device, or if the M-SEARCH request matches a device type or service type supported by the device. Multi-homed devices shall send the search response using the same UPnP-enabled interface on which the search request was received. The URL specified in the LOCATION field value shall specify an address that is reachable on that interface.

Devices responding to a multicast M-SEARCH should wait a random period of time between 0 seconds and the number of seconds specified in the MX field value of the search request before responding, in order to avoid flooding the requesting control point with search responses from multiple devices. If the search request results in the need for a multiple part response from the device, those multiple part responses should be spread at random intervals through the time period from 0 to the number of seconds specified in the [MX](#) header field. Devices are allowed to assume an MX field value less than that specified in the [MX](#) header field. If the [MX](#) header field specifies a field value greater than 5, the device should assume that it contained the value 5 or less. Devices shall not stop responding to other requests while waiting the random delay before sending a response.

For multicast M-SEARCH requests, if the search request does not contain an [MX](#) header field, the device shall silently discard and ignore the search request. If the [MX](#) header field specifies a field value greater than 5, the device should assume that it contained the value 5 or less.

For multicast M-SEARCH requests, if the search request does contain the TCP:PORT.UPNP.ORG header field, the device shall reply on the TCP port indicated in the M-SEARCH request, however it does not have to spread and repeat the required messages since the transport over TCP is reliable, hence ignoring the MX value. The reply to the control point can be formatted as 1 message replying to all applicable USN as required by the M-SEARCH syntax. The list of USNs can be conveyed by a comma separated list, see RFC 2616. Hence using this option will reduce the number of messages sent as responses to the M-SEARCH and will speed up the detection of devices in the network.

Any device responding to a unicast M-SEARCH should respond within 1 second.

The URL specified in the LOCATION header field of the [M-SEARCH](#) response shall be reachable by the control point to which the response is directed.

Responses to [M-SEARCH](#) requests are intentionally parallel to advertisements, and as such, follow the same pattern as listed for [NOTIFY](#) with [ssdp:alive](#) (above) except that instead of the [NT](#) header field there is an [ST](#) header field here. The response shall be sent in the following format. Values in *italics* are placeholders for actual values.

```
HTTP/1.1 200 OK
CACHE-CONTROL: max-age = seconds until advertisement expires
DATE: when response was generated
EXT:
LOCATION: URL for UPnP description for root device
SERVER: OS/version UPnP/2.0 product/version
ST: search target
USN: composite identifier for the advertisement
BOOTID.UPNP.ORG: number increased each time device sends an initial announce or an update message
CONFIGID.UPNP.ORG: number used for caching description information
SEARCHPORT.UPNP.ORG: number identifies port on which device responds to unicast M-SEARCH
```



Note: No body is present in a response to a request with method [M-SEARCH](#), but note that the message shall have a blank line following the last header field.

(Note: No need to limit TTL for the IP packet in response to a search request.)

Listed below are details for the header fields appearing in the listing above. Field names are not case sensitive. All field values are case sensitive except where noted.

## Response line

Shall be "HTTP/1.1 200 OK"

## Header fields

### CACHE-CONTROL

Required. Field value shall have the max-age directive ("max-age=") followed by an integer that specifies the number of seconds the advertisement is valid. After this duration, control points should assume the device (or service) is no longer available; as long as a control point has received at least one advertisement that is still valid from a root device, any of its embedded devices or any of its services, then the control point can assume that all are available. The number of seconds should be greater than or equal to 1800 seconds (30 minutes), although exceptions are defined in the text above. Specified by UPnP vendor. Other directives shall not be sent and shall be ignored when received.

### DATE

Recommended. Field value contains date when response was generated. "rfc1123-date" as defined in RFC 2616.

### EXT

Required for backwards compatibility with UPnP 1.0. (Header field name only; no field value.)

### LOCATION

Required. Field value contains a URL to the UPnP description of the root device. Normally the host portion contains a literal IP address rather than a domain name in unmanaged networks. Specified by UPnP vendor. Single absolute URL (see RFC 3986).

### SERVER

Required. Specified by UPnP vendor. String. Field value shall begin with the following "product tokens" (defined by HTTP/1.1). The first product token identifies the operating system in the form *OS name/OS version*, the second token represents the UPnP version and shall be [UPnP/2.0](#), and the third token identifies the product using the form *product name/product version*. For example, "SERVER: *unix/5.1* [UPnP/2.0](#) *MyProduct/1.0*".

### ST

Required. Field value contains Search Target. Single URI. The response sent by the device depends on the field value of the ST header field that was sent in the request. In some cases, the device shall send multiple response messages as follows. If the received ST field value was:

#### [ssdp:all](#)

Respond  $3+2d+k$  times for a root device with  $d$  embedded devices and  $s$  embedded services but only  $k$  distinct service types (see clause 1.1.2, "SSDP message header fields" for a definition of each message to be sent). Field value for ST header field shall be the same as for the NT header field in NOTIFY messages with [ssdp:alive](#). (See above.)

#### [upnp:rootdevice](#)

Respond once for root device. Shall be [upnp:rootdevice](#).

#### uuid:*device-UUID*

Respond once for each matching device, root or embedded. Shall be uuid:*device-UUID* where *device-UUID* is specified by the UPnP vendor. See clause 1.1.4, "UUID format and recommended generation algorithms" for the MANDATORY UUID format.

urn:[schemas-upnp-org:device:deviceType:ver](#)

Respond once for each matching device, root or embedded. Shall be urn:[schemas-upnp-org:device:deviceType:ver](#) where [deviceType](#) and [ver](#) are defined by UPnP Forum working committee and [ver](#) shall contain the version of the device type contained in the M-SEARCH request.

urn:[schemas-upnp-org:service:serviceType:ver](#)

Respond once for each matching service type. shall be urn:[schemas-upnp-org:service:serviceType:ver](#) where [serviceType](#) and [ver](#) are defined by the UPnP Forum working committee and [ver](#) shall contain the version of the service type contained in the M-SEARCH request.

urn:[domain-name:device:deviceType:ver](#)

Respond once for each matching device, root or embedded. shall be urn:[domain-name:device:deviceType:ver](#) where [domain-name](#) (a Vendor Domain Name), [deviceType](#) and [ver](#) are defined by the UPnP vendor and [ver](#) shall contain the version of the device type from the M-SEARCH request. Period characters in the Vendor Domain Name shall be replaced with hyphens in accordance with RFC 2141.

urn:[domain-name:service:serviceType:ver](#)

Respond once for each matching service type. shall be urn: [domain-name:service:serviceType:ver](#) where [domain-name](#) (a Vendor Domain Name), [serviceType](#) and [ver](#) are defined by the UPnP vendor and [ver](#) shall contain the version of the service type from the M-SEARCH request. Period characters in the Vendor Domain Name shall be replaced with hyphens in accordance with RFC 2141.

## USN

Required. Field value contains Unique Service Name. Identifies a unique instance of a device or service. shall be one of the following. (See Table 1 1, “Root device discovery messages”, Table 1 2, “Embedded device discovery messages”, and Table 1 3, “Service discovery messages” above.) The prefix (before the double colon) shall match the value of the UDN element in the device description OR the lower ST version used in the M-Search request. (Section 2, “Description” explains the UDN element.) Single URI.

uuid:[device-UUID::upnp:rootdevice](#)

Sent once for root device where [device-UUID](#) is specified by UPnP vendor. See section 1.1.4, “UUID format and Recommended generation algorithms” for the MANDATORY UUID format.

uuid:[device-UUID](#)

Sent once for every device, root or embedded, where device-UUID is specified by the UPnP vendor. See section 1.1.4, “UUID format and RECOMMENDED generation algorithms” for the MANDATORY UUID format.

uuid:[device-UUID::urn:schemas-upnp-org:device:deviceType:ver](#)

Sent once for every device, root or embedded, where device-UUID is specified by the UPnP vendor, deviceType and ver are defined by UPnP Forum working committee and ver specifies version of the device type. See section 1.1.4, “UUID format and RECOMMENDED generation algorithms” for the MANDATORY UUID format.

uuid:[device-UUID::urn:schemas-upnp-org:service:serviceType:ver](#)

Sent once for every service where device-UUID is specified by the UPnP vendor, serviceType and ver are defined by UPnP Forum working committee and ver specifies version of the device type. See section 1.1.4, “UUID format and RECOMMENDED generation algorithms” for the MANDATORY UUID format.

uuid:[device-UUID::urn:domain-name:device:deviceType:ver](#)

Sent once for every device, root or embedded, where device-UUID, domain-name (a Vendor Domain Name), deviceType and ver are defined by the UPnP vendor and ver specifies the version of the device type. See section 1.1.4, “UUID format and RECOMMENDED generation algorithms” for the MANDATORY UUID format. Period characters in the Vendor Domain Name shall be replaced by hyphens in accordance with RFC 2141.

uuid:[device-UUID::urn:domain-name:service:serviceType:ver](#)

Sent once for every service where device-UUID, domain-name (a Vendor Domain Name), serviceType and ver are defined by the UPnP vendor and ver specifies the version of the service type. See section 1.1.4, “UUID format and recommended generation algorithms” for the MANDATORY UUID format. Period characters in the Vendor Domain Name shall be replaced by hyphens in accordance with RFC 2141.

#### BOOTID.UPNP.ORG

Required. As defined in clause 1.2, and 1.2.2.

#### CONFIGID.UPNP.ORG

Allowed. As defined in clause 1.2, and 1.2.2.

#### SEARCHPORT.UPNP.ORG

Allowed. As defined in clause 1.2, and 1.2.2.

#### SECURELOCATION.UPNP.ORG

Allowed. As defined in clause 1.2.2.

If there is an error with the search request (such as an invalid field value in the MAN header field, a missing MX header field, or other malformed content), the device shall silently discard and ignore the search request; sending of error responses is PROHIBITED due to the possibility of packet storms if many devices send an error response to the same request.

## 1.4 References

RFC 2141, URN Syntax. Available at: <http://www.ietf.org/rfc/rfc2141.txt>.

RFC 2616, HTTP: Hypertext Transfer Protocol 1.1. Available at: <http://www.ietf.org/rfc/rfc2616.txt>.

RFC 2774, HTTP Extension Framework. Available at: <http://www.ietf.org/rfc/rfc2774.txt>.

RFC 3986, Uniform Resource Identifiers (URI): Generic Syntax. Available at: <http://www.ietf.org/rfc/rfc3986.txt>.

RFC 4340, Datagram Congestion Control Protocol (DCCP). Available at: <http://www.ietf.org/rfc/rfc4340.txt>.

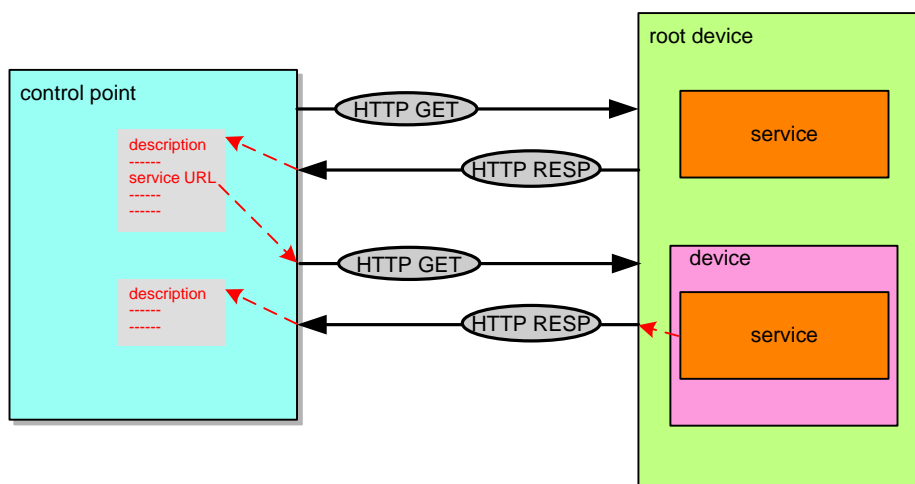
[1] DCE variant of Universal Unique Identifiers (UUIDs), The Open group, 1997, Available at: <http://www.opengroup.org/onlinepubs/9629399/apdxa.htm>.

## 2 Description

*Description is Step 2 in UPnP networking. Description comes after addressing (Step 0) where devices get a network address, and after discovery (Step 1) where control points find interesting device(s). Description enables control (Step 3) where control points send commands to device(s), eventing (Step 4) where control points listen to state changes in device(s), and presentation (Step 5) where control points may display an html user interface for device(s).*

After a control point has discovered a device, the control point still knows very little about the device -- only the information that was in the discovery message, i.e., the device's (or service's) UPnP type, the device's universally-unique identifier, and a URL to the device's UPnP description. For the control point to learn more about the device and its capabilities, or to interact with the device, the control point shall retrieve a description of the device and its capabilities from the URL provided by the device in the discovery message.

**Figure 2-1: — Description architecture**



The UPnP description for a device is partitioned into two logical parts: a *device description* describing the physical and logical containers, and *service descriptions* describing the capabilities exposed by the device. A UPnP device description includes vendor-specific manufacturer information like the model name and number, serial number, manufacturer name, URLs to vendor-specific Web sites, etc. (details below). For each service included in the device, the device description lists the service type, service name, a URL for a service description, a URL for control, and a URL for eventing. A device description also includes a description of all embedded devices and a URL for presentation of the aggregate. This clause explains UPnP device descriptions, and the clauses on Control, Eventing, and Presentation explain how URLs for control, eventing, and presentation are used respectively.

Note that a single physical device is allowed to include multiple logical devices. Multiple logical devices can be modeled as a single root device with embedded devices (and services) or as multiple root devices (perhaps with no embedded devices). In the former case, there is one UPnP device description for the root device, and that device description contains a description for all embedded devices. In the latter case, there are multiple UPnP device descriptions, one for each root device.

A UPnP device description is written by a UPnP vendor. The description is in XML syntax and is usually based on a standard UPnP Device Template. A UPnP Device Template is produced by a UPnP Forum working committee; they derive the template from the UPnP Device Schema, which was derived from standard constructions in XML. This clause explains the format for a UPnP device description, UPnP Device Templates, and the part of the UPnP Device Schema that covers devices.

A UPnP service description includes a list of commands, or *actions*, to which the service responds, and parameters, or *arguments* for each action. A service description also includes a list of variables. These variables model the state of the service at run time, and are described in terms of their data type, range, and event characteristics. This clause explains the description of actions, arguments, state variables, and the properties of those variables. The clause on Eventing explains event characteristics.

Like a UPnP device description, a UPnP service description is written by a UPnP vendor. The description is in XML syntax and is usually based on a standard UPnP Service Template. A UPnP Service Template is produced by a UPnP Forum working committee; they derived the template from the UPnP Service Schema, augmenting it with human language where necessary. The UPnP Service Schema is derived using the conventions of XML Schema. This clause explains the format for a UPnP service description, UPnP Service Templates, typical augmentations in human language, and the part of the UPnP Service Schema that covers services.

UPnP vendors can differentiate their devices by extending services (see clause 2.7, “Non-standard vendor extensions and limitations”), including additional UPnP services, or embedding additional devices. When a control point retrieves a particular device's description, these added features are exposed to the control point for control and eventing. The device and service descriptions authoritatively document the implementation of the device.

Retrieving a UPnP device description is simple: the control point issues an HTTP GET request on the URL in the discovery message, and the device returns the device description. Retrieving a UPnP service description is a similar process that uses a URL within the device description. The protocol stack, method, header fields, and body for the response and request are explained in detail below. Description documents shall be sent using the same IP address on which the HTTP GET request was received.

As long as *at least one* of the discovery advertisements from a root device, any of its embedded devices or any of its services have not expired and *none* of the advertisements have been cancelled, a control point is allowed to assume that the root device and all its embedded devices and all its services are available. The device and service descriptions are allowed to be retrieved at any point since the device and service descriptions are static as long as the device and its services are available. If a device cancels *at least one* of its advertisements or if *all* the advertisements expire, a control point should assume the device and its services are no longer available. If a device needs to change one of these descriptions, it shall cancel its outstanding advertisements and re-advertise. Consequently, control points should not assume that device and service descriptions are unchanged if a device re-appears on the network, but they can detect whether descriptions changed if a changed CONFIGID.UPNP.ORG field value is present in the announcements.

Like discovery, description plays an important role in the interoperability of devices and control points using different versions of UPnP networking. As explained in clause 1, “Discovery”, the UPnP Device Architecture is versioned with both a major and a minor version. The major version and minor version are separate integer numbers; they are not to be interpreted or compared as though they were a single decimal number, even though they are allowed to appear as such in print. Advances in minor versions shall be a compatible superset of earlier minor versions of the same major version; therefore device vendors are free to implement standardized devices and services on versions of the architecture with a higher minor version number. Advances in major version are not required to be supersets of earlier versions and are not guaranteed to be backward compatible. However UDA version 2.0 is specified as a superset of UDA 1.1 and is thus backwards compatible with UDA 1.x versions. Therefore UDA 2.0 control points shall maintain interoperability with UDA 1.x devices. UDA 1.x control points can work with UDA 2.0 devices, but can't access the additional functionality specified in UDA 2.0. The architecture version of a root device, all its embedded devices and all its services shall be the same. Version information is communicated in description messages as a backup to the information communicated in discovery messages. This clause explains the format of version information in description messages.

Device and service types standardized by UPnP Forum working committees or created by vendors have an integer version. Every later version of a device or service shall be a fully backwardly compatible superset of the previous version, i.e., compared to earlier versions of the device, it shall include all mandatory embedded devices and services of the same or later version. The UPnP device or service type remains the same across all versions of a device whereas the device or service version shall be larger for later versions. Versions of device and service templates are allowed to have non-integer versions (such as “0.9”) during development in the working committee, but this shall become an integer upon standardization. Devices and services are allowed to have a version number greater than the major version number of the architecture they are designed for (e.g., “Power:2” is allowed to be designed to work on UDA version 1.0); there is no direct correlation between the version of a device or service template and the architecture version with which it is designed to work. If a non-backward-compatible version of a device or service is defined, it shall have a different device or service name to indicate that it is not backwardly compatible (and version numbers of the new type shall restart at 1).

UPnP device and service types are “building blocks” that is allowed to be assembled in various combinations. Both standard and vendor-defined device types are allowed to be embedded in standard device types. Both standard and vendor-defined device types are allowed to be embedded in vendor-defined device types. Likewise, both standard and vendor-defined service types are allowed to be embedded in both standard and vendor-defined device types. A control point that is capable of operating with a particular device or service type shall at least recognize that device or service type even when it is embedded within another device type (standard or vendor-defined) that it does not recognize. For example, if a standard service type “Print:1” is defined, and a standard device type “Printer:1” is defined that contains the “Print:1” service, a control point that wishes to use the “Print:1” service shall find and use it whether the service is embedded within a “urn:schemas-upnp-org:device:Printer:1” device or embedded within a vendor-defined “urn:acme-com:device:Printer:1” or “urn:acme-com:device:AcmeMultifunctionPrinter:1” device.

The remainder of this clause first explains how devices are described, explaining details of vendor-specific information, embedded devices, and URLs for control, eventing, and presentation. Second, it explains UPnP Device Templates. Third, it explains how services are described, explaining details of actions, arguments, state variables, and properties of those variables. Then it explains UPnP Service Templates, and the UPnP Service Schema. Finally, this clause explains in detail how a control point retrieves device and service descriptions from a device.

## 2.1 Generic requirements on HTTP usage

This subclause defines generic requirements on HTTP usage in UPnP Version 2.0. HTTP is the underlying transport for:

- Description (see clause 2, “Description”)
- Control (see clause 3, “Control”)
- Eventing (see clause 4, “Eventing”)
- Presentation (clause 5, “Presentation”)

The baseline transport for all devices and control points is recommended to be HTTP/1.1 compliant (as defined in RFC 2616) but at least shall be HTTP/1.0 compliant (as defined in RFC 1945). Vendors are free to implement and Working Committees are free to require for new device classes implementations of more recent versions of HTTP that are backwards compatible with HTTP version 1.0, such as HTTP version 1.1 as defined in RFC 2616. However whatever version is implemented, all required components defined by the specified HTTP version shall be implemented.

If a control point uses an HTTP/1.0 binding on a SOAP request without setting the KeepAlive token, the device shall close the socket after responding. If a control point uses an HTTP/1.1 binding on a SOAP request, and sets the “Connection:CLOSE” token, the device shall close the socket after responding.

### USER-AGENT header field

Control points can add the USER-AGENT header field to any UPnP-related HTTP request to signal that they support UPnP 1.1. Working Committees are allowed to require presence of this header on description retrieval, action invocations and event subscriptions for newly defined services.

```
USER-AGENT: OS/version UPnP/2.0 product/version
```

#### USER-AGENT

Allowed. Specified by UPnP vendor. String. Field value shall begin with the following “product tokens” (defined by HTTP/1.1). The first product token identifies the operating system in the form *OS name/OS version*, the

second token represents the UPnP version and shall be [UPnP/2.0](#), and the third token identifies the product using the form *product name/product version*. For example, "USER-AGENT: *unix/5.1* [UPnP/2.0](#) *MyProduct/1.0*".

### Accept-Encoding header field

Control points can add the Accept-Encoding header field to any UPnP-related HTTP request to signal that they support the type of encoding.

```
Accept-Encoding: compress, gzip
```

#### Accept-Encoding

Allowed. Specified by RFC2616 section 14.3. Allowed encoding are identity, compress and gzip. The identity encoding shall be present and without value q=0. When the request is satisfied by the Server, then the content-encoding header shall be present. The value of the content\_encoding header specifies the used encoding type of the response. Used values are vendor specific.

### UPnP friendly Name header field

Control points shall add the CPFN.UPNP.ORG header field to any UPnP-related HTTP request to signal the friendly name of the control point.

```
CPFN.UPNP.ORG: friendly name
```

#### CPFN.UPNP.ORG

Required. Specifies the friendly name of the control point. The friendly name is vendor specific. When Device Protection is implemented the CPFN.UPNP.ORG shall be the same as the <Name> of Device Protection unless the Device Protection <Alias> is defined, in which case it shall use the <Alias>.

### UPnP identifier header field

Control points can add the CPUUID.UPNP.ORG header field to any UPnP-related HTTP request to signal the unique identifier of the control point.

```
CPUUID.UPNP.ORG: uuid of control point
```

#### CPUUID.UPNP.ORG

Allowed. uuid of the control point. When the control point is implemented in a UPnP device it is recommended to use the UDN of the co-located UPnP device. When implemented, all specified requirements for uuid usage in devices also apply for control points. See section 1.1.4. Note that when Device Protection is implemented the CPUUID.UPNP.ORG shall be the same as the uuid used in Device Protection.

### Vendor-defined or working committee-defined HTTP Header fields

HTTP field names defined by vendors or working committees shall have the following format:

field-name = token "." domain-name

where the domain-name shall be a Vendor Domain Name or shall be "UPNP.ORG" (for working committee defined field names), and in addition shall satisfy the token format as defined in RFC 2616 clause 2.2. Field names are case-insensitive.

### HTTP/1.0 Persistent connections

Some implementations of HTTP/1.0 defined what is known as persistent connections. There are many practical uses for this functionality, as it may reduce overhead for a given device by allowing resources to be used more efficiently. However, this functionality for HTTP/1.0 is not officially defined in the specification and classified as experimental. Further, the way it has been experimentally defined is flawed in such a way that it may cause sessions to hang in certain scenarios. This functionality shall not be implemented by any UPnP devices or control points that implement HTTP version 1.0.

## **HTTP/1.0 HEAD request**

Some implementations utilize the HEAD request to try to predetermine the amount of memory required to process a GET request. Some servers may not know that size of the content because it may be dynamic. In such cases, the responses will not contain a CONTENT-LENGTH header field. As such, control points shall not rely on the CONTENT-LENGTH header field being specified for a HEAD response.

## **HTTP/1.1 General**

When a device or control point implements HTTP/1.1, all requirements of HTTP/1.0 shall be maintained, with the exception of the CONTENT-LENGTH header field, which shall not be specified when doing chunked transfers.

## **HTTP status codes**

Servers shall return appropriate HTTP status codes for invalid requests. A device or control point shall use a 4xx HTTP status code for responses that indicate a problem with the format of a request or response. For example, if an HTTP client makes a PUT request to a server that does not implement the PUT method, the server should return a "405 Method not Allowed" HTTP status code and shall return a 4xx series HTTP status code. Another example is if an HTTP client makes a request to a server that is malformed HTTP or not well formed XML, the server should return a "400 Bad Request" HTTP status code and shall return a 4xx series HTTP status code. While clients are not required to understand specific status codes, they shall understand classes of status codes. For example, a 4xx series HTTP status code signifies an improper request, whereas a 5xx series HTTP status code signifies a processing error for a valid request.

## **HTTP/1.1 and HTTP/1.0 compatibility**

Devices and control points that implement HTTP/1.1 shall be able to interoperate with HTTP/1.0 control points and devices. Care shall be taken when devices and control points process requests, such that the response generated is compatible with the HTTP version specified in the request. For example, if an HTTP/1.0 request is made, the device or control point shall not return an HTTP/1.1 chunked response.

## **HTTP/1.1 HOST header field and use of the HOST header field with HTTP/1.0**

The 'HOST' header field shall be specified in all requests, because HTTP/1.1 allows support for virtual domains, which rely on this header field to determine the target destination.

The HOST header field shall also be included in HTTP/1.0 requests, for backwards compatibility with UPnP 1.0, which REQUIRES the HOST header field to be present without explicitly mentioning a HTTP version.

## **HTTP/1.1 EXPECT: 100-Continue**

Servers are allowed to send a "100-Continue" HTTP status code to let the client know that the header fields received have been processed. If a client will rely on this status response before sending the body, it shall send the "EXPECT: 100-Continue" header field in the request. If a server received this header field in the request, it shall not wait for the request body before sending the continue response. However, a client shall be prepared to handle cases when the "EXPECT: 100-Continue" header field is not sent, but a "100-Continue" HTTP status code is still received from the server.

## **HTTP/1.1 Chunked Encoding**

Devices and control points that advertise support for HTTP/1.1 shall have support for decoding chunked encoded messages. Chunked encoded messages are allowed to contain Chunk-Extensions, which are delineated with a ';'. Extensions that are not recognized shall be ignored, which includes the absence of an extension, but the presence of the delineator.

Chunked encoding also allows responses and requests to include trailer fields, which are header fields that follow the body. Devices and control points shall only send trailer fields if



the request contained the 'TE' header field (indicates trailer processing is supported), or if the trailer fields in the response only contain allowed metadata that can be safely ignored.

Before a control point uses chunked encoding to make a request to a device, it shall check to ensure that the device is an HTTP/1.1 device. Devices are allowed to use different HTTP engines (that support different versions) for description, control, eventing and presentation. Therefore, to correctly identify which HTTP version is used for processing control requests, a HEAD request is allowed to be issued to the corresponding control URL.

### HTTP/1.1 Persistent Connections

Persistent connections is the default behavior defined by HTTP/1.1. It is strongly recommended that this behavior be maintained, as it may be beneficial in many scenarios, as it allows for resources to be utilized more efficiently. Support for Pipelined request handling is also recommended if persistent connections are supported.

If a server responds with a "CONNECTION: close" header line, it shall close the session after responding. Similarly if a client specifies "CONNECTION: close" in the request, the server shall also close the session after responding.

When Requests are pipelined to a server, the server shall answer the requests in the order that they are received. Clients shall also be prepared to retry connections if pipelining fails, for example, if the server does not support them.

### HTTP/1.1 Redirect restrictions

HTTP/1.1 defines allowed support for redirecting an HTTP request. UPnP 2.0 devices are allowed to redirect a request, although this is not recommended. If a UPnP 2.0 device redirects a request, it shall respond with a "307 Temporary Redirect" HTTP status code (see also RFC 2616). UPnP 2.0 devices shall not return any other HTTP/1.1 redirect options. Control points shall implement HTTP/1.1 redirect and should redirect the request upon receiving a "307 Temporary Redirect" HTTP status code (see also RFC 2616).

## 2.2 Generic requirements on XML usage

XML namespace prefixes do not have to be the specific strings that are used in the examples in this specification. They can be any value that obeys the rules of the general XML namespace mechanism as outlined in the *Namespaces in XML* specification. Devices shall accept requests that use other legal XML namespace prefixes.

If an XML element has no value (i.e. it contains the empty string), it is valid to combine the opening and closing XML tags (e.g., "<actionname/>" instead of "<actionname></actionname>").

## 2.3 Device description

The UPnP description for a device contains several pieces of vendor-specific information, definitions of all embedded devices, URL for presentation of the device, and listings for all services, including URLs for control and eventing. In addition to defining non-standard devices (which is allowed to contain both vendor-defined and standard embedded devices and services), UPnP vendors are allowed to add embedded devices and services to standard devices. To illustrate these, below is a listing with placeholders (in *italics>*) for actual elements and values. Some of these placeholders would be specified by a UPnP Forum working committee (colored *red*) or by a UPnP vendor (colored *purple*). For a non-standard device, all of these placeholders would be specified by a UPnP vendor. Elements defined by the UPnP Device Architecture are colored *green*. Immediately following the listing is a detailed explanation of the elements, attributes, and values.

```
<?xml version="1.0"?>
<root xmlns="urn:schemas-upnp-org:device-1-0"
  configId="configuration number">
  <specVersion>
```

```

    <major>2</major>
    <minor>0</minor>
  </specVersion>
  <device>
    <deviceType>urn:schemas-upnp-org:device:deviceType:v</deviceType>
    <friendlyName>short user-friendly title</friendlyName>
    <manufacturer>manufacturer name</manufacturer>
    <manufacturerURL>URL to manufacturer site</manufacturerURL>
    <modelDescription>long user-friendly title</modelDescription>
    <modelName>model name</modelName>
    <modelName>model name</modelName>
    <modelNumber>model number</modelNumber>
    <modelURL>URL to model site</modelURL>
    <serialNumber>manufacturer's serial number</serialNumber>
    <UDN>uuid:UUID</UDN>
    <UPC>Universal Product Code</UPC>
    <iconList>
      <icon>
        <mimetype>image/format</mimetype>
        <width>horizontal pixels</width>
        <height>vertical pixels</height>
        <depth>color depth</depth>
        <url>URL to icon</url>
      </icon>
      <!-- XML to declare other icons, if any, go here -->
    </iconList>
    <serviceList>
      <service>
        <serviceType>urn:schemas-upnp-org:service:serviceType:v</serviceType>
        <serviceId>urn:upnp-org:serviceId:serviceID</serviceId>
        <SCPDURL>URL to service description</SCPDURL>
        <controlURL>URL for control</controlURL>
        <eventSubURL>URL for eventing</eventSubURL>
      </service>
      <!-- Declarations for other services defined by a UPnP Forum working committee
           (if any) go here -->
      <!-- Declarations for other services added by UPnP vendor (if any) go here -->
    </serviceList>
    <deviceList>
      <!-- Description of embedded devices defined by a UPnP Forum working committee
           (if any) go here -->
      <!-- Description of embedded devices added by UPnP vendor (if any) go here -->
    </deviceList>
    <presentationURL>URL for presentation</presentationURL>
  </device>
</root>

```

Listed below are details for each of the elements, attributes, and values appearing in the listing above. All elements and attributes are case sensitive; HTTP specifies case sensitivity for URLs; other values are not case sensitive except where noted. The order of elements is significant. Except where noted: required elements shall occur exactly once (no duplicates), and recommended or allowed elements are allowed to occur at most once. Note that some implementations are allowed to strictly enforce the length limits for various elements noted below, and therefore working committees are advised to heed all limits specified.

<?xml>

Required for all XML documents. Case sensitive.

<root>

Required. Shall have “urn:schemas-upnp-org:device-1-0” as the value for the `xmlns` attribute; this references the UPnP Device Schema (described below). Case sensitive. Has the following attribute:

`configId`

Required. Specifies the configuration number to which the device description belongs. See clause 1, “Discovery” for further definition and usage of the configuration number.

Contains all other elements describing the root device, i.e., contains the following child elements:

<specVersion>

Required. In device templates, defines the lowest version of the architecture on which the device can be implemented. In actual UPnP devices, defines the architecture on which the device is implemented. Contains the following sub elements:

<major>

Required. Major version of the UPnP Device Architecture. Shall be 2 for devices implemented on a UPnP 2.0 architecture.

<minor>

Required. Minor version of the UPnP Device Architecture. Shall be 0 for devices implemented on a UPnP 2.0 architecture. Shall accurately reflect the version number of the UPnP Device Architecture supported by the device.

<URLBase>

Use of URLBase is deprecated from UPnP 1.1 onwards; UPnP 2.0 devices shall NOT include URLBase in their description documents. For full definition of URLBase, see the UPnP 1.0 specification.

<device>

Required. Contains the following sub elements:

<deviceType>

Required. UPnP device type. Single URI.

- For standard devices defined by a UPnP Forum working committee, shall begin with "urn:[schemas-upnp-org:device:](#)" followed by the standardized device type suffix, a colon, and an integer device version i.e. urn:[schemas-upnp-org:device:deviceType:ver](#). The highest supported version of the device type shall be specified.
- For non-standard devices specified by UPnP vendors, shall begin with "urn:", followed by a Vendor Domain Name, followed by ":", followed by a device type suffix, colon, and an integer version, i.e., "urn:[domain-name:device:deviceType:ver](#)". Period characters in the Vendor Domain Name shall be replaced with hyphens in accordance with RFC 2141. The highest supported version of the device type shall be specified.

The device type suffix defined by a UPnP Forum working committee or specified by a UPnP vendor shall be <= 64 chars, not counting the version suffix and separating colon.

<friendlyName>

Required. Short description for end user. Is allowed to be localized (see ACCEPT-LANGUAGE and CONTENT-LANGUAGE header fields). Specified by UPnP vendor. String. Should be < 64 characters.

<manufacturer>

Required. Manufacturer's name. Is allowed to be localized (see ACCEPT-LANGUAGE and CONTENT-LANGUAGE header fields). Specified by UPnP vendor. String. Should be < 64 characters.

<manufacturerURL>

Allowed. Web site for Manufacturer. Is allowed to have a different value depending on language requested (see ACCEPT-LANGUAGE and CONTENT-LANGUAGE header fields). Specified by UPnP vendor. Single URL.

<modelDescription>

Recommended. Long description for end user. Is allowed to be localized (see ACCEPT-LANGUAGE and CONTENT-LANGUAGE header fields). Specified by UPnP vendor. String. Should be < 128 characters.

<modelName>

Required. Model name. Is allowed to be localized (see ACCEPT-LANGUAGE and CONTENT-LANGUAGE header fields). Specified by UPnP vendor. String. Should be < 32 characters.

<modelName>

Recommended. Model number. Is allowed to be localized (see ACCEPT-LANGUAGE and CONTENT-LANGUAGE header fields). Specified by UPnP vendor. String. Should be < 32 characters.

<modelURL>

Allowed. Web site for model. Is allowed to have a different value depending on language requested (see ACCEPT-LANGUAGE and CONTENT-LANGUAGE header fields). Specified by UPnP vendor. Single URL.

<serialNumber>

Recommended. Serial number. Is allowed to be localized (see ACCEPT-LANGUAGE and CONTENT-LANGUAGE header fields). Specified by UPnP vendor. String. Should be < 64 characters.

<UDN>

Required. Unique Device Name. Universally-unique identifier for the device, whether root or embedded. shall be the same over time for a specific device instance (i.e., shall survive reboots). shall match the field value of the NT header field in device discovery messages. shall match the prefix of the USN header field in all discovery messages. (Clause 1, "Discovery" explains the NT and USN header fields.) shall begin with "uuid:" followed by a UUID suffix specified by a UPnP vendor. See clause 1.1.4, "UUID format and recommended generation algorithms" for the MANDATORY UUID format.

<UPC>

Allowed. Universal Product Code. 12-digit, all-numeric code that identifies the consumer package. Managed by the Uniform Code Council. Specified by UPnP vendor. Single UPC.

<iconList>

Required if and only if device has one or more icons. Specified by UPnP vendor. Contains the following sub elements:

<icon>

Recommended. Icon to depict device in a control point UI. Is allowed to have a different value depending on language requested (see ACCEPT-LANGUAGE and CONTENT-LANGUAGE header fields). Icon sizes to support are vendor-specific. Contains the following sub elements:

<mimetype>

Required. Icon's MIME type (see RFC 2045, 2046, and 2387). Single MIME image type. At least one icon should be of type "image/png" (Portable Network Graphics, see IETF RFC 2083).

<width>

Required. Horizontal dimension of icon in pixels. Integer.

<height>

Required. Vertical dimension of icon in pixels. Integer.

<depth>

Required. Number of color bits per pixel. Integer.

<url>

Required. Pointer to icon image. (XML does not support direct embedding of binary data. See note below.) Retrieved via HTTP. Shall be relative to the URL at which the device description is located in accordance with clause 5 of RFC 3986. Specified by UPnP vendor. Single URL.

<serviceList>

Allowed. Contains the following sub elements:

<service>

Allowed. Repeated once for each service defined by a UPnP Forum working committee. If UPnP vendor differentiates device by adding additional, standard UPnP services, repeated once for each additional service. Contains the following sub elements:

<serviceType>

Required. UPnP service type. Shall not contain a hash character (#, 23 Hex in UTF-8). Single URI.

- For standard service types defined by a UPnP Forum working committee, shall begin with “urn:[schemas-upnp-org:service:](#)” followed by the standardized service type suffix, colon, and an integer service version i.e. urn:[schemas-upnp-org:device:serviceType:ver](#). The highest supported version of the service type shall be specified.
- For non-standard service types specified by UPnP vendors, shall begin with “urn:”, followed by a Vendor Domain Name, followed by “:[service:](#)”, followed by a service type suffix, colon, and an integer service version, i.e., “urn:[domain-name:service:serviceType:ver](#)”. Period characters in the Vendor Domain Name shall be replaced with hyphens in accordance with RFC 2141. The highest supported version of the service type shall be specified.

The service type suffix defined by a UPnP Forum working committee or specified by a UPnP vendor shall be <= 64 characters, not counting the version suffix and separating colon.

<serviceId>

Required. Service identifier. Shall be unique within this device description. Single URI.

- For standard services defined by a UPnP Forum working committee, shall begin with “urn:[upnp-org:serviceId:](#)” followed by a service ID suffix i.e. urn:[upnp-org:serviceId:serviceID](#). If this instance of the specified service type (i.e. the <[serviceType](#)> element above) corresponds to one of the services defined by the specified device type (i.e. the <[deviceType](#)> element above), then the value of the service ID suffix shall be the service ID defined by the device type for this instance of the service. Otherwise, the value of the service ID suffix is vendor defined. (Note that [upnp-org](#) is used instead of [schemas-upnp-org](#) in this case because an XML schema is not defined for each service ID.)
- For non-standard services specified by UPnP vendors, shall begin with “urn:”, followed by a Vendor Domain Name, followed by “:[serviceId:](#)”, followed by a service ID suffix, i.e., “urn:[domain-name:serviceId:serviceID](#)”. If this instance of the specified service type (i.e. the <[serviceType](#)> element above) corresponds to one of the services defined by the specified device type (i.e. the <[deviceType](#)> element above), then the value of the service ID suffix shall be the service ID defined by the device type for this instance of the service. Period characters in the Vendor Domain Name shall be replaced with hyphens in accordance with RFC 2141.

The service ID suffix defined by a UPnP Forum working committee or specified by a UPnP vendor shall be <= 64 characters.

<SCPURL>

Required. URL for service description. (See clause 2.5, “Service description” below.) shall be relative to the URL at which the device description is located in accordance with clause 5 of RFC 3986. Specified by UPnP vendor. Single URL.

<controlURL>

Required. URL for control (see clause 3, “Control”). shall be relative to the URL at which the device description is located in accordance with clause 5 of RFC 3986. Specified by UPnP vendor. Single URL.

<eventSubURL>

Required. URL for eventing (see clause 4, “Eventing”). shall be relative to the URL at which the device description is located in accordance with clause 5 of RFC 3986. shall be unique within the device; any two services shall not have the same URL for eventing. If the service has no evented variables, this element shall be present but shall be empty (i.e., <eventSubURL></eventSubURL>.) Specified by UPnP vendor. Single URL.

<deviceList>

Required if and only if root device has embedded devices. Contains the following sub elements:

<device>

Required. Repeat once for each embedded device defined by a UPnP Forum working committee. If UPnP vendor differentiates device by embedding additional UPnP devices, repeat once for each embedded device. Contains sub elements as defined above for root sub element device.

<presentationURL>

Recommended. URL to presentation for device (see clause 5, "Presentation"). shall be relative to the URL at which the device description is located in accordance with the rules specified in clause 5 of RFC 3986. Specified by UPnP vendor. Single URL.

Control points should recognize and interoperate with services using `serviceld` values other than the value defined by the device type. If multiple instances of a service exist, control points should by default (unless directed otherwise by user action) use the service instance associated with the `serviceld` value defined by the device type. If none of the instances of the service have the `serviceld` value defined by the device type, the control point may use any service instance. When only one instance of the service exists, control points should use that instance even if the `serviceld` value does not match that defined by the device type.

For future extensibility and according to the requirements in clause 2.7, "Non-standard vendor extensions" and clause 2.8, "UPnP Device Schema", when processing XML like the listing above, devices and control points shall ignore: (a) any unknown elements and their sub elements or content, and (b) any unknown attributes and their values.

Subject to the constraints defined in clause 2.7, "Non-standard vendor extensions" and clause 2.8, "UPnP Device Schema", control points and devices shall ignore any XML comments or XML processing instructions embedded in UPnP device and service descriptions that they do not understand. UPnP device descriptions shall be encoded using UTF-8.

When the value of any text element or attribute contains one or more characters reserved as markup (such as ampersand ("&") or less than ("<")), the text shall be escaped in accordance with the provisions of clause 2.4 of the XML specification and each such character replaced with the equivalent numeric representation or string (such as "&amp;" or "&lt;"). Such characters appearing in URLs are allowed to also be percent-encoded in accordance with the URL percent-encoding rules specified in clauses 2.1 and 2.4 of RFC 3986.

XML does not support directly embedding binary data, e.g., icons in UPnP device descriptions. Binary data are allowed to be converted into text (and thereby embedded into XML) using an XML data type of either `bin.base64` (a MIME-style base 64 encoding for binary data) or `bin.hex` (hexadecimal digits represent octets). Alternatively, the data can be passed indirectly, as it were, by embedding a URL in the XML and transferring the data in response to a separate HTTP request; the icon(s) in UPnP device descriptions are transferred in this latter manner.

If any icons are included, at least one should be in the Portable Network Graphics (PNG) format defined in RFC 2083, indicated by the MIME type "image/png", and not use progressive encoding. No specific icon sizes are recommended due to the wide variety preferred by various control points; control point vendors are encouraged to publish implementation guidelines.

The use of `URLBase` element is deprecated by this specification. UPnP 2.0 devices shall not include `URLBase` in their description documents. To ensure interoperability with UPnP 1.0 devices, control points shall be able to process `URLBase` if it is specified and use it for resolving relative URLs that appear elsewhere in the description. If relative URLs are included in the device description, control points shall resolve them into absolute URLs in accordance with clause 5 of RFC 3986, using either `URLBase` (if specified) or the location from which the device description was retrieved as the base URL, before using these URLs for their respective purposes.

Note that in version 1.0 of the UPnP Device Architecture, the `serviceList` element was required, and it was required to contain at least one `service` element. These requirements

were subsequently rescinded to accommodate the InternetGatewayDevice:1 and Basic:1 device types. If the device has no services, the **serviceList** element is allowed to be omitted entirely, or it is allowed to be present but contain no **service** elements.

## 2.4 UPnP Device Template

The listing above also illustrates the relationship between a UPnP device description and a UPnP Device Template. As explained above, the UPnP device description is written by a UPnP vendor, in XML, following a UPnP Device Template. A UPnP Device Template is produced by a UPnP Forum working committee as a means to standardize devices.

By appropriate specification of placeholders, the listing above can be either a UPnP Device Template or a UPnP device description. Recall that some placeholders would be defined by a UPnP Forum working committee (colored *red*), i.e., the UPnP device type identifier, required UPnP services, and required UPnP embedded devices (if any). If these were defined, the listing would be a UPnP Device Template, codifying the standard for this type of device. UPnP Device Templates are one of the key deliverables from UPnP Forum working committees.

Taking this another step further, the remaining placeholders in the listing above would be specified by a UPnP vendor (colored *purple*), i.e., vendor-specific information. If these placeholders were specified (as well as the others), the listing would be a UPnP device description, suitable to be delivered to a control point to enable control, eventing, and presentation.

Put another way, the UPnP Device Template defines the overall type of device, and each UPnP device description instantiates that template with vendor-specific information. The first is created by a UPnP Forum working committee; the latter, by a UPnP vendor.

## 2.5 Service description

The UPnP description for a service defines actions and their arguments, and state variables and their data type, range, and event characteristics.

Each service shall have zero or more actions. Each action shall have zero or more arguments. Each argument is designated as either an input or an output argument. Input arguments shall be listed first. If an action has one or more output arguments, the first output argument is allowed to be marked as a return value. Each argument shall correspond to one of the <stateVariable> elements in the <serviceStateTable> in the SCPD.

Each service shall have one or more state variables.

In addition to defining non-standard services, UPnP vendors are allowed to add actions and services to standard devices, and are allowed to embed standard services and devices in non-standard devices.

To illustrate these points, below is a listing with placeholders (in *italics*) for actual elements and values. For a standard UPnP service, some of these placeholders would be defined by a UPnP Forum working committee (colored *red*) or specified by a UPnP vendor (*purple*). For a non-standard service, all of these placeholders would be specified by a UPnP vendor. Elements defined by the UPnP Device Architecture are colored *green*. Immediately following the listing is a detailed explanation of the elements, attributes, and values.

```
<?xml version="1.0"?>
<scpd
  xmlns="urn:schemas-upnp-org:service-1-0"
  xmlns:dt1="urn:domain-name:more-datatypes"
  <!-- Declarations for other namespaces added by UPnP Forum working committee (if any) go
        here -->
  <!-- The value of the attribute shall remain as defined by the UPnP Forum working
        committee. -->
  xmlns:dt2="urn:domain-name:vendor-datatypes"
  <!-- Declarations for other namespaces added by UPnP vendor (if any) go here -->
```

```

<!-- Vendors shall change the URN's domain-name to a Vendor Domain Name -->
<!-- Vendors shall change vendor-datatypes to reference a vendor-defined namespace -->
configId="configuration number">
  <specVersion>
    <major>2</major>
    <minor>0</minor>
  </specVersion>
  <actionList>
    <action>
      <name>actionName</name>
      <argumentList>
        <argument>
          <name>argumentNameIn1</name>
          <direction>in</direction>
          <relatedStateVariable>stateVariableName</relatedStateVariable>
        </argument>
        <!-- Declarations for other IN arguments defined by UPnP Forum working
             Committee (if any) go here -->
        <argument>
          <name>argumentNameOut1</name>
          <direction>out</direction>
          <retval/>
          <relatedStateVariable>stateVariableName</relatedStateVariable>
        </argument>
        <argument>
          <name>argumentNameOut2</name>
          <direction>out</direction>
          <relatedStateVariable>stateVariableName</relatedStateVariable>
        </argument>
        <!-- Declarations for other OUT arguments defined by UPnP Forum working
             committee (if any) go here -->
      </argumentList>
    </action>
    <!-- Declarations for other actions defined by UPnP Forum working committee
         (if any) go here -->
    <!-- Declarations for other actions added by UPnP vendor (if any) go here -->
  </actionList>
  <serviceStateTable>
    <stateVariable sendEvents="yes"|"no" multicast="yes"|"no">
      <name>variableName</name>
      <dataType>basic data type</dataType>
      <defaultValue>default value</defaultValue>
      <allowedValueRange>
        <minimum>minimum value</minimum>
        <maximum>maximum value</maximum>
        <step>increment value</step>
      </allowedValueRange>
    </stateVariable>
    <stateVariable sendEvents="yes"|"no" multicast="yes"|"no">
      <name>variableName</name>
      <dataType type="dt1:variable data type">string</dataType>
      <defaultValue>default value</defaultValue>
      <allowedValueList>
        <allowedValue>enumerated value</allowedValue>
        <!-- Other allowed values defined by UPnP Forum working committee
             (if any) go here -->
        <!-- Other allowed values defined by vendor (if any) go here -->
      </allowedValueList>
    </stateVariable>
    <stateVariable sendEvents="yes"|"no" multicast="yes"|"no">
      <name>variableName</name>
      <dataType type="dt2:vendor data type">string</dataType>
      <defaultValue>default value</defaultValue>
    </stateVariable>
    <!-- Declarations for other state variables defined by UPnP Forum working committee
         (if any) go here -->
    <!-- Declarations for other state variables added by UPnP vendor (if any) go here -
->
  </serviceStateTable>
</scpd>

```

Listed below are details for each of the elements, attributes, and values appearing in the listing above. All elements and attributes (including action, argument, and state variable



names) are case sensitive; values are not case sensitive except where noted. Except where noted, required elements shall occur exactly once (no duplicates), and recommended or Allowed elements are allowed to occur at most once.

<?xml>

Required for all XML documents. Case sensitive.

<scpd>

Required. Shall have "urn:[schemas-upnp-org:service-1-0](#)" as the value for the `xmlns` attribute; this references the UPnP Service Schema (explained below). Case sensitive. Has the following attribute:

`configId`

Required. Specifies the configuration number to which the service description belongs. See clause 1, "Discovery" for further definition and usage of the configuration number.

Contains all other elements describing the service, i.e., contains the following sub elements:

<specVersion>

Required. In service templates, defines the lowest version of the architecture on which the service can be implemented. In actual UPnP services, defines the architecture on which the service is implemented. Contains the following sub elements:

<major>

Required. Major version of the UPnP Device Architecture. Shall be 2 for services implemented on a UPnP 2.0 architecture.

<minor>

Required. Minor version of the UPnP Device Architecture. Shall be 0 for services implemented on a UPnP 2.0 architecture. Shall accurately reflect the version number of the UPnP Device Architecture supported by the device.

<actionList>

Required if and only if the service has actions. (Each service is allowed to have  $\geq 0$  actions.) Contains the following sub element(s):

<action>

Required. Repeat once for each action defined by a UPnP Forum working committee. If UPnP vendor differentiates service by adding additional actions, repeat once for each additional action. Contains the following sub elements:

<name>

Required. Name of action. Shall not contain a hyphen character ("-", 2D Hex in UTF-8) nor a hash character ("#", 23 Hex in UTF-8). Case sensitive. First character shall be a USASCII letter ("A"- "Z", "a"- "z"), USASCII digit ("0"- "9"), an underscore ("\_"), or a non-experimental Unicode letter or digit greater than U+007F. Succeeding characters shall be a USASCII letter ("A"- "Z", "a"- "z"), USASCII 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.

- For standard actions defined by a UPnP Forum working committee, Shall not begin with "X" nor "A".
- For non-standard actions specified by a UPnP vendor and added to a standard service, shall begin with "X\_", followed by a Vendor Domain Name, followed by the underscore character ("\_"), followed by the vendor-assigned action name. The vendor-assigned action name shall comply with the syntax rules defined above.

String. Should be < 32 characters.

<argumentList>

Required if and only if parameters are defined for action. (Each action is allowed to have  $\geq 0$  parameters.) Contains the following sub element(s):

<argument>

Required. Repeat once for each parameter. UPnP vendors shall not add vendor-defined arguments to actions defined by a UPnP Forum working committees. Contains the following sub elements:

<name>

Required. Name of formal parameter. The name should be chosen to reflect the semantic use of the argument. shall not contain a hyphen character (“-”, 2D Hex in UTF-8). First character shall be a USASCII letter (“A”-“Z”, “a”-“z”), USASCII digit (“0”-“9”), an underscore (“\_”), or a non-experimental Unicode letter or digit greater than U+007F. Succeeding characters shall be a USASCII letter (“A”-“Z”, “a”-“z”), USASCII 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. String. Case sensitive. Should be < 32 characters.

<direction>

Required. Defines whether argument is an input or output parameter. shall be either “in” or “out” and not both. All input arguments shall be listed before any output arguments.

<retval>

Allowed. Identifies at most one output argument as the return value. If included, shall be included as a subelement of the first output argument. (Element only; no value.)

<relatedStateVariable>

Required. shall be the name of a state variable. Case Sensitive. Defines the type of the argument; see further explanation below in this clause.

<serviceStateTable>

Required. (Each service shall have => 1 state variables.) Contains the following sub element(s):

<stateVariable>

Required. Repeat once for each state variable defined by a UPnP Forum working committee. If UPnP vendor differentiates service by adding additional state variables, repeat once for each additional state variable. Has the following attributes:

sendEvents

Allowed. Defines whether event messages will be generated when the value of this state variable changes. Default value is “yes”. Non-evented state variables shall set this attribute to “no”.

- For standard state variables defined by a UPnP Forum working committee, the working committee decides whether the variable is evented and the value of the sendEvents attribute shall not be altered by a vendor.
- For non-standard state variables specified by a UPnP vendor and added to a standard service, the vendor is allowed to decide whether the non-standard state variable will be evented or not.

multicast

Allowed. Defines whether event messages will be delivered using multicast eventing. Default value is “no”. If the multicast is set to “yes”, then all events sent for this state variable shall be unicast AND multicast.

- For standard state variables defined by a UPnP Forum working committee, the working committee decides whether the state variable is multicast and the value of the multicast attribute shall not be altered by a vendor.
- For non-standard variables specified by a UPnP vendor and added to a standard service, the vendor is allowed to decide whether the non-standard variable will be delivered using multicast eventing.

The <stateVariable> element contains the following sub elements:

<name>

Required. Name of state variable. Shall not contain a hyphen character (“-”, 2D Hex in UTF-8). First character shall be a USASCII letter (“A”-“Z”, “a”-“z”), USASCII digit (“0”-“9”), an underscore (“\_”), or a non-experimental Unicode letter or digit greater than U+007F. Succeeding characters shall be a USASCII letter (“A”-“Z”, “a”-“z”), USASCII 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. Case sensitive.

- For standard state variables defined by a UPnP Forum working committee, shall not begin with “X” nor “A”.
- For non-standard state variables specified by a UPnP vendor and added to a standard service, shall begin with “X”, followed by a Vendor Domain Name, followed by the underscore character (“\_”), followed by the vendor-assigned state variable name. The vendor-assigned state variable name shall comply with the syntax rules defined above.

String. Should be < 32 characters.

<dataType>

Required. Same as data types defined by XML Schema, Part 2: Datatypes. Defined by a UPnP Forum working committee for standard state variables; specified by UPnP vendor for extensions. Has an allowed `type` attribute:

`type`

Allowed. If the `type` attribute is present, the value of the <dataType> element shall be “string”. The value of the `type` attribute overrides the “string” value; it defines the data type using a fully qualified data type name according to the conventions of XML schema and can refer to XML Schema simple types, service-local complex types and service-local extended simple types. Service-local data types are defined in a corresponding UPnP Service Template or they are allowed to be vendor-specific. See also clause 2.5.1, “Defining and processing extended data types” and clause 2.5.2, “String equivalents of extended data types”.

For example: <dataType type="xsd:byte">string</dataType>

For a state variable using an extended data type via the `type` attribute and containing complex data, the <defaultValue>, <allowedValueList> and <allowedValueRange> elements shall not be present. In such case the restrictions for the data type shall be described in the data type schema provided in the service template document.

The <dataType> element shall have one of the following values:

`ui1`

Unsigned 1 Byte int. Same format as int without leading sign.

`ui2`

Unsigned 2 Byte int. Same format as int without leading sign.

`ui4`

Unsigned 4 Byte int. Same format as int without leading sign.

`ui8`

Unsigned 8 Byte int. Same format as int without leading sign.

`i1`

1 Byte int. Same format as int.

`i2`

2 Byte int. Same format as int.

`i4`

4 Byte int. Same format as int. shall be between -2147483648 and 2147483647.

i8

8 Byte int. Same format as int. shall be between -9,223,372,036,854,775,808 and 9,223,372,036,854,775,807, from  $-(2^{63})$  to  $2^{63} - 1$ .

int

Fixed point, integer number. Is allowed to have leading sign. Is allowed to have leading zeros, which should be ignored by the recipient. (No currency symbol.) (No grouping of digits to the left of the decimal, e.g., no commas.)

r4

4 Byte float. Same format as float. shall be between  $3.40282347E+38$  to  $1.17549435E-38$ .

r8

8 Byte float. Same format as float. shall be between  $-1.79769313486232E308$  and  $-4.94065645841247E-324$  for negative values, and between  $4.94065645841247E-324$  and  $1.79769313486232E308$  for positive values, i.e., IEEE 64-bit (8-Byte) double.

number

Same as r8.

fixed.14.4

Same as r8 but no more than 14 digits to the left of the decimal point and no more than 4 to the right.

float

Floating point number. Mantissa (left of the decimal) and/or exponent is allowed to have a leading sign. Mantissa and/or exponent Is allowed to have leading zeros, which should be ignored by the recipient. Decimal character in mantissa is a period, i.e., whole digits in mantissa separated from fractional digits by period (“.”). Mantissa separated from exponent by “E”. (No currency symbol.) (No grouping of digits in the mantissa, e.g., no commas.)

char

Unicode string. One character long.

string

Unicode string. No limit on length.

date

Date in a subset of ISO 8601 format without time data.

dateTime

Date in ISO 8601 format with allowed time but no time zone.

dateTime.tz

Date in ISO 8601 format with allowed time and allowed time zone.

time

Time in a subset of ISO 8601 format with no date and no time zone.

time.tz

Time in a subset of ISO 8601 format with allowed time zone but no date.

boolean

“0” for false or “1” for true. The values “true”, “yes”, “false”, or “no” are deprecated and shall not be sent but shall be accepted when received. When received, the values “true” and “yes” shall be interpreted as true and the values “false” and “no” shall be interpreted as false.

bin.base64

MIME-style Base64 encoded binary BLOB. Takes 3 Bytes, splits them into 4 parts, and maps each 6 bit piece to an octet. (3 octets are encoded as 4.) No limit on size.

bin.hex

Hexadecimal digits representing octets. Treats each nibble as a hex digit and encodes as a separate Byte. (1 octet is encoded as 2.) No limit on size.

uri

Universal Resource Identifier.

uuid

Universally Unique ID. See clause 1.1.4, "UUID format and recommended generation algorithms" for the MANDATORY UUID format.

<defaultValue>

Recommended. Expected, initial value. Defined by a UPnP Forum working committee or delegated to UPnP vendor. shall match data type. shall satisfy <allowedValueList> or <allowedValueRange> constraints. For a state variable using an extended data type via the `type` attribute and containing complex data, the <defaultValue> element shall not be present.

<allowedValueList>

Recommended. Enumerates legal string values. PROHIBITED for data types other than string. At most one of the <allowedValueRange> or <allowedValueList> elements are allowed to be specified. Sub elements are ordered. For a state variable using an extended data type via the `type` attribute and containing complex data, the <allowedValueList> element shall not be present. Contains the following sub elements:

<allowedValue>

Required. A legal value for a string variable. Defined by a UPnP Forum working committee for standard state variables; if the UPnP Forum working committee permits it, UPnP vendors are allowed to add vendor-specific allowed values to standard state variables. Specified by UPnP vendor for extensions. String. shall be < 32 characters.

<allowedValueRange>

Recommended. Defines bounds for legal numeric values; defines resolution for numeric values. Defined only for numeric data types (i.e. integers and floats). At most one of the <allowedValueRange> or <allowedValueList> elements are allowed to be specified. For a state variable using an extended data type via the `type` attribute and containing complex data, the <allowedValueRange> element shall not be present. Contains the following sub elements which shall have the same type as the state variable:

<minimum>

Required. Inclusive lower bound. Defined by a UPnP Forum working committee or delegated to UPnP vendor. Single numeric value. The value of the <minimum> element shall be less than the value of the <maximum> element. If a working committee has assigned an explicit value for this element, then vendors shall use that value. Otherwise, vendors shall choose their own value, but always within the allowed range for the data type of this state variable. If the working committee defines an allowed range for this state variable, then the value shall be within that allowed range as defined by the <step> value (See below).

<maximum>

Required. Inclusive upper bound. Defined by a UPnP Forum working committee or delegated to UPnP vendor. Single numeric value. The value of the <maximum> element shall be greater than the value of the <minimum> element. If a working committee has assigned an explicit value for this element, then vendors shall use that value. Otherwise, vendors shall choose their own value, but always within the allowed range for the data type of this state variable. If the working committee defines an allowed range for this

state variable, then the value shall be within that allowed range as defined by the <step> value (See below).

<step>

Recommended. Defines the set of allowed values permitted for the state variable between the <minimum> and <maximum>. The value of the <step> element divides the inclusive range from <minimum> value to <maximum> value into an integral number of equal parts. Additionally, <maximum> value = <minimum> value + n \* <step> value, where n is a positive integer. Defined by a UPnP Forum working committee or delegated to UPnP vendor. If a working committee has assigned an explicit value for this element, then vendors shall use that value. Otherwise, vendors are allowed to choose their own value. When the <step> element is omitted and the data type of the state variable is an integer, the default value of step is 1; otherwise, when step is omitted, the state variable are allowed to be set to any value within the inclusive range of <minimum> value to <maximum> value. Single numeric value.

Note that one should be careful when dealing with floating point values so that conversions and/or rounding errors do not cause inaccurate comparison operations.

The <relatedStateVariable> element of an <argument> element definition shall be the name of a state variable defined in the same service description. The <relatedStateVariable> element defines the *data type* of the argument; there is not necessarily any semantic relationship between an argument and the related state variable used to define its type. The <relatedStateVariable> element shall specify the name of a state variable in the service state table which has the same data type, allowed value list, or allowed value range as the argument. If no state variable exists with an appropriate definition, the working committee (or vendor) shall define an additional state variable for that purpose; state variables which are defined solely for the purpose of describing the type of an argument shall have a name that includes the prefix "A\_ARG\_TYPE\_".

The <allowedValueList> and <allowedValueRange> elements are allowed to be used to indicate optional device capabilities. Working committees are allowed to REQUIRE all values in the list or range to be supported by all vendors (no options), REQUIRE a minimum subset with additional values being allowed, or allow vendors to entirely decide which portions of the list or range to support. Vendors are allowed to add additional, vendor-specific values to the allowed value list by using the "x" prefix on the vendor-defined allowed values, if permitted by working committees. However, it is be noted that greater flexibility in allowed capabilities reduces the number of values that control points can depend on to be present, with corresponding impacts on interoperability. If device capabilities are expected to change during device operation, working committees should define evented state variables or separate actions to detect device capabilities rather than embedding capabilities information in the service description, because the latter requires cancellation of advertisements and readvertisement each time the service description document is changed. If the service description is used to convey capabilities information, the device shall omit from the service description any allowed elements (actions, allowed values, etc.) that are not implemented.

For a state variable using an extended data type via the *type* attribute and containing complex data, the <defaultValue>, <allowedValueList> and <allowedValueRange> elements shall not be present. In such case the restrictions for the data type shall be described in the data type schema typically provided in the service template document.

### 2.5.1 Defining and processing extended data types

The optional *type* attribute of the <dataType> element as defined in clause 2.5, "Service description" above allows a service description document (SCPD) to include extended data types (defined by the UPnP technical committee, a UPnP working committee or vendor-specific data types) that have more structure and expression than the existing XSD data types. As mentioned above, this *type* attribute can only be applied when the base data type is of type *string*. The value attached to the *type* attribute refers to a data type from a separate schema defined outside this document.

As a first RECOMMENDATION on the use of extended data types, if UPnP actions only have simple arguments, these should be declared using UPnP defined data types, instead of XML schema simple types. This enables use of such UPnP actions by UPnP 1.0 stacks that are not XML-schema enabled.

As a further RECOMMENDATION on extended data types that are allowed to be defined, arrays should be declared by using a sequence with an element type of which the number of occurrences is restricted. For example, if an array-type “myArrayType” of 50 long integers needs to be declared, this could be the corresponding schema fragment:

```
<xsd:complexType name="myArrayType">
  <xsd:sequence>
    <xsd:element name="x" type="xsd:long" minOccurs="50" maxOccurs="50"/>
  </xsd:sequence>
</xsd:complexType>
```

References to this type (as with any XML namespace) can be made in one of two ways. The first option is a fully qualified namespace reference in the `type` attribute alone. In this case the namespace reference in the `type` attribute shall not only refer to the schema, but also to the type within that schema.

```
<scpd xmlns="urn:schemas-upnp-org:service-1-0"
  configId="configuration number">
  ...
  <dataType type="urn:domain-name:schema-name:datatype-name">
    string
  </dataType>
  ...
</scpd>
```

The second option is to define the namespace at the beginning of the SCPD document and then make a reference in the type definition. In this case, the `type` attribute contains a prefix that identifies the namespace, followed by the data type-name.

```
<scpd xmlns="urn:schemas-upnp-org:service-1-0"
  xmlns:dt="urn:domain-name:schema-name"
  configId="configuration number">
  ...
  <dataType type="dt:datatype-name">
    string
  </dataType>
  ...
</scpd>
```

Implementations shall support both formats. The first format is potentially easier to parse, while the second format may result in shorter description files (i.e. when the same namespace is used multiple times in the same document).

These data types written in XSD Schema need not be processed at run-time. Instead, an implementer is allowed to use the referred schema as a standard description of the type to parse for that particular type attribute. To allow run-time schema processing of extended data types, an optional location of the extended data type schema is allowed to be expressed in the SCPD using the standard XSD `xsi:schemaLocation` and `xsi:noNamespaceSchemaLocation` attributes as defined in clause 4.3.2 of XML Schema Part 1. These attributes can be used in the root SCPD element (essentially an instance document) where the extended data type is defined, as illustrated below:

```

<scpd xmlns="urn:schemas-upnp-org:service-1-0"
  xmlns:dt="urn:domain-name:schema-name"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:domain-name:schema-name
    http://some.company.com/dir/file.xsd"
  configId="configuration number">
  ...
  <dataType type="dt:datatype-name">
    string
  </dataType>
  ...
</scpd>

```

Alternatively, these attributes are allowed to be declared on use in the <dataType> element where the existing fully qualified namespace and type name for the extended data type are defined. An example for reference is given below:

```

<scpd xmlns="urn:schemas-upnp-org:service-1-0"
  configId="configuration number">
  ...
  <dataType type="urn:domain-name:schema-name:datatype-name"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="http://some.company.com/dir/file.xsd">
    string
  </dataType>
  ...
</scpd>

```

## 2.5.2 String equivalents of extended data types

A number of working committees have created services based on UPnP 1.0 (which does not support extended data types) that define their own encoding of information inside specific string-type state variables. To provide these working committees with an upgrade path to extended datatypes written in native XML<sup>3</sup>, a mechanism is defined that gives working committees the *option* to define the “string equivalent” of an extended data type (working committees can decide not to). If a string equivalent of an extended data type is defined, there are two valid ways to represent the value of the data type: either as an extended data type, written in native XML, or as a string, that encodes the datatype as specified by the working committee.

The mechanism uses the USER-AGENT header field in action invocations and event subscriptions (see also clause 2.1, “Generic requirements on HTTP usage”). If a control point invokes an action without a USER-AGENT header field, or if the USER-AGENT header field does not specify UPnP version 2.0 or greater, the values of in-arguments and out-arguments shall be encoded using the “string equivalent”.

If a control point invokes an action with a USER-AGENT header field that specifies UPnP version 2.0 or greater, the values of in-arguments and out-arguments shall be encoded using the extended data type written in native XML.

If a control point has subscribed to events without a USER-AGENT header field, or if the USER-AGENT header field specifies a UPnP version less than 2.0, all values of complex-type evented state variables that are sent to the control point shall be encoded using the “string equivalent”. If no “string equivalent” is defined for an evented state variable, subscription without the correct USER-AGENT header field shall be refused.

---

<sup>3</sup> In this text “native XML” refers to datatypes formatted according to XML-schema using the normal XML format, while “string-equivalent of an extended datatype” refers to encoding a complex data type inside a UPnP string, examples of which (escaped XML, comma-separated lists) can be found in the ContentDirectory:1 specification.



If a control point has subscribed to events with a USER-AGENT header field that specifies UPnP version 2.0 or greater, all values of complex-type evented state variables that are sent to the control point shall be encoded using the extended data type written in native XML.

### 2.5.3 Generic requirements

For future extensibility and according to the requirements in clause 2.7, “Non-standard vendor extensions” and clause 2.8, “UPnP Device Schema”, when processing XML like the listing above, devices and control points shall ignore: (a) any unknown elements and their sub elements or content, and (b) any unknown attributes and their values.

Subject to the constraints defined in clause 2.7, “Non-standard vendor extensions” and clause 2.8, “UPnP Device Schema”, control points and devices shall ignore any XML comments or XML processing instructions embedded in UPnP device and service descriptions that they do not understand. UPnP service descriptions shall be encoded using UTF-8.

When the value of any text element or attribute contains one or more characters reserved as markup (such as ampersand (“&”) or less than (“<”)), the text shall be escaped in accordance with the provisions of clause 2.4 of the XML specification and each such character replaced with the equivalent numeric representation or string (such as “&amp;” or “&lt;”). Such characters appearing in URLs are allowed to also be percent-encoded in accordance with the URL percent-encoding rules specified in clauses 2.1 and 2.4 of RFC 3986. Note that it is logically possible for a service to have no actions but have state variables and eventing; though unlikely, such a service would be an autonomous information source. However, a service with no state variables is PROHIBITED.

Unlike device descriptions, service descriptions and associated values shall not use locale-specific values; this includes service descriptions, values of action arguments, and values of state variables. Instead, most action arguments and state variables shall use values that are expressed in a locale-independent manner; control points are allowed to convert and/or format the information from a standard form into the correct language and/or format for the locale. For example, dates are represented in a locale-independent format (ISO 8601), and integers are represented without locale-specific formatting (e.g., no currency symbol, no grouping of digits). String values shall be represented in a locale-independent manner. Variables with an allowedValueList shall use token values in the language of UPnP standards and not reflect strings intended to be displayed in a localized user interface.

### 2.5.4 Ordering of Elements

The order of XML elements in device and service description documents shall adhere to the order as defined in the corresponding specification as defined by the working committee for that device or service type. Furthermore, the order of elements (e.g. arguments) in control messages and in their responses shall adhere to the order defined in the device’s service description document.

Note: UPnP 1.0 does NOT REQUIRE that the order of XML elements in device and service description documents adheres to the order as defined in the corresponding schema (as defined by the working committee) for that device or service type. However, it *does* REQUIRE that control messages and responses are ordered according to the corresponding device’s service description, a REQUIREMENT that is sometimes overlooked. Therefore, when receiving messages from UPnP 1.0 services, control points should be able to process out-of-order elements; and when transmitting messages to UPnP 1.0 services, control points shall send elements in the order defined by that particular device’s service description.

### 2.5.5 Versioning

Services standardized by UPnP Forum working committees have an integer version. Every later version of a service shall be a superset of the previous version, i.e., it shall include all actions and state variables exactly as they are defined by earlier versions of the service. The UPnP service type remains the same across all versions of a service whereas the service version shall be larger for later versions. Versions of device and service templates are

allowed to have non-integer versions (such as “0.9”) during development in the working committee, but this shall become an integer upon standardization. Devices and services are allowed to have a version number greater than the major version number of the architecture they are designed for (e.g., “Power:2” may be designed to work on UDA version 1.0).

## 2.6 UPnP Service Template

The listing above also illustrates the relationship between a UPnP service description and a UPnP Service Template. As explained above, the UPnP description for a service is written by a UPnP vendor, in XML, following a UPnP Service Template. A UPnP Service Template is produced by a UPnP Forum working committee as a means to standardize devices.

By appropriate specification of placeholders, the listing above can be either a UPnP Service Template or a UPnP service description. Recall that some placeholders would be defined by a UPnP Forum working committee (colored *red*), i.e., actions and their parameters, and states and their data type, range, and event characteristics. If these were specified, the listing above would be a UPnP Service Template, codifying the standard for this type of service. Along with UPnP Device Templates (see clause 2, “Description”), UPnP Service Templates are one of the key deliverables from UPnP Forum working committees.

Taking this another step further, the remaining placeholders in the listing above would be specified by a UPnP vendor (colored *purple*), i.e., additional, vendor-specified actions and state variables. If these placeholders were specified (as well as the others), the listing would be a UPnP service description, suitable for effective control of the service within a device.

Put another way, the UPnP Service Template defines the overall type of service, and each UPnP service description instantiates that template with vendor-specific additions. The first is created by a UPnP Forum working committee; the latter by a UPnP vendor.

## 2.7 Non-standard vendor extensions and limitations

As explained above, UPnP vendors are allowed to differentiate their devices and extend a standard device by including additional services and embedded devices. Similarly, UPnP vendors are allowed to extend a standard service by including additional actions, state variables or allowed values. Naming conventions and conditions for each of these are listed in the table below and explained in detail above.

**Table 2-1: — Vendor extensions**

| Type of extension  | Standard   | Non-Standard  |
|--|--|---|
| device type  | urn:schemas-upnp-org:device:deviceType:v   | urn:domain-name:device:deviceType:v   |
| service type   | urn:schemas-upnp-org:service:serviceType:v   | urn:domain-name:service:serviceType:v   |
| service ID   | urn:upnp-org:serviceId:serviceID   | urn:domain-name:serviceId:serviceID   |
| action name  | Shall comply with the syntax rules of the standardized action name as defined in clause 2.5, “Service description”.        | Shall comply with the syntax rules of the non-standardized action name as defined in clause 2.5, “Service description”.   |
| stateVariable name   | Shall comply with the syntax rules of the standardized stateVariable name as defined in clause 2.5, “Service description”. | Shall comply with the syntax rules of the non-standardized stateVariable name as defined in clause 2.5, “Service description”.  |
| allowedValue value   | Shall be a legal value for a string variable. Only values explicitly defined by a working committee are allowed.           | Permitted only if allowed by the working committee. Shall begin with a Vendor Domain Name, followed by the underscore character (“_”), followed by a legal value for a string variable. |
| XML elements and their attributes in device or service description | Defined by the UPnP Device and Service Schemas.  | Arbitrary XML, scoped by one or more XML namespaces owned by the vendor. Shall be enclosed in an element that begins with “X_”.   |

| Type of extension  | Standard  | Non-Standard  |
|--|---|---|
| XML attributes of standard elements in device or service description | Defined by the UPnP Device and Service Schemas. | Arbitrary attributes, scoped by one or more XML namespaces, owned by the vendor. Shall begin with “ <u>X</u> ”. |

As the last two rows of the table above indicate, UPnP vendors are allowed to also add non-standard XML to 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 are allowed to be added to standard elements provided these attributes are scoped by a vendor-owned XML namespace and begin with “X”.

To illustrate this, below are listings with placeholders (in *italics>) for actual elements and values. Some of these placeholders would be specified by a UPnP vendor (*purple*) and some are defined by the UPnP Device Architecture (*green*).*

```
<RootStandardElement
  xmlns="urn:schemas-upnp-org:device-1-0"
  xmlns:n="domain-name:schema-name">
  <!-- other XML -->
  <AnyStandardElement n:X_VendorAttribute="arbitrary string value">
  <!-- other XML -->
  </AnyStandardElement>
  <!-- other XML -->
</RootStandardElement>
```

<RootStandardElement>

A standard root element. `xmlns` attribute defines namespaces, in this case, a standard UPnP namespace and a non-standard namespace with the prefix *n*. (Note: *n* is just a placeholder. A vendor can specify any prefix to identify the namespace that is valid according to the *Namespaces in XML* specification.)

- For device descriptions, shall be <root>.
- For service descriptions, shall be <scpd>.

<AnyStandardElement>

Any standard element, root or otherwise, content of text or element only. Shall already be included as part of the standard device or service description. X\_VendorAttribute shall begin with “X”. (Prefix “A” is reserved.) are allowed to have an arbitrary string value.

```
<StandardComplexType n:X_VendorAttribute="vendor value">
  <n:X_VendorElement xmlns:n="domain-name:schema-name">
  <!-- arbitrary XML -->
  </n:X_VendorElement>
</StandardComplexType>
```

<StandardComplexType>

Element of complex type. Shall already be included as part of the standard device or service description.

- For device descriptions, shall be one of: <root>, <specVersion>, <device>, <iconList>, <icon>, <serviceList>, <service>, or <deviceList>.
- For service descriptions, shall be one of: <scpd>, <actionList>, <action>, <argumentList>, <argument>, <serviceStateTable>, <stateVariable>, <allowedValueList>, or <allowedValueRange>.

<X\_VendorElement>

Shall begin with “X”. (Prefix “A” is reserved.) Shall have a value for the `xmlns` attribute. Is allowed to contain arbitrary XML.

### 2.7.1 Placement of Additional Elements and Attributes

Instances of any UPnP schema, including device and service descriptions, control actions, errors and event notifications, are allowed to include additional XML elements (other than those defined by the UPnP Forum) *only* at the end of an ordered sequence of elements

corresponding to a given complex type. Additionally, instances of any UPnP schema are allowed to include additional attributes with any element.

#### **Exception for UPnP 1.0 devices:**

UPnP 1.0 allows the inclusion of additional elements anywhere within device and service descriptions, control actions, errors and event notifications, provided that the XML is well-formed. Therefore, when receiving messages from UPnP 1.0 devices, control points shall handle unknown elements and attributes found anywhere within the message.

### **2.8 UPnP Device Schema**

The paragraphs above explain UPnP device descriptions and illustrate how one would be instantiated from a UPnP Device Template. As explained, UPnP Device Templates are produced by UPnP Forum working committees, and these templates are based upon the UPnP Device Schema. This schema defines the structures and data types used to create UPnP Device Templates. clause B.1, “UPnP Device Schema” contains the schema; below is an explanation of this schema.

The UPnP Device Schema is written in XML and according to the conventions of XML Schema (Part 1: Structures, Part 2: Datatypes). XML Schema provides a method of describing the structure of an XML document. The XML Schema description language itself is based upon XML. The language is very robust; it specifies which elements are required vs. allowed, element nesting, data types for values (as well as other properties not of interest here) and much more. The UPnP Device Schema uses these XML Schema constructions to define elements like <specVersion>, <URLBase>, <deviceType>, etc., listed in detail above. Because the UPnP Device Schema is constructed using a precise description language, it is unambiguous. As the UPnP Device Schema, UPnP Device Templates, and UPnP device descriptions are all machine-readable, software tools are allowed to be devised to validate the latter two, checking that they contain all the required elements, are correctly nested, and have values of the correct data types.

### **2.9 UPnP Service Schema**

The paragraphs above explain UPnP service descriptions and illustrate how one would be instantiated from a UPnP Service Template. Like UPnP Device Templates, UPnP Service Templates are produced by UPnP Forum working committees, and these templates are based upon the UPnP Service Schema. This schema defines the structure and data types used to create UPnP Service Templates. As explained above, the UPnP Service Schema is written in XML according to the conventions of XML Schema (Part 1: Structures, Part 2: Datatypes). clause B.2, “UPnP Service Schema” contains a listing of this schema

### **2.10 UPnP Datatype Schema**

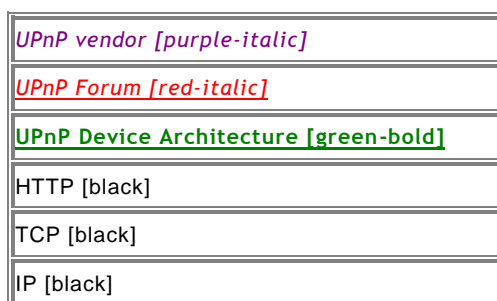
The UPnP basic data types for state variables are defined in clause 2.5, “Service description”. For any extended data types for state variables used by a service template, the service template shall include either a reference to all relevant schemas for the extended data types or include a new service specific datatype schema with a corresponding unique target namespace. If any extended data types are used for state variables within an SCPD, the corresponding namespace for each extended data type shall be referenced within the SCPD according to the “Namespaces in XML” specification. Clause 2.5, “Service description” contains an example SCPD with namespace declarations.

### **2.11 Retrieving a description using HTTP**

As explained above, after a control point has discovered a device, it still knows very little about the device. To learn more about the device and its capabilities, the control point shall retrieve the UPnP description for the device using the URL provided by the device in the discovery message. Then, the control point shall retrieve one or more service descriptions using the URL(s) provided in the device description. This is a simple HTTP-based process and uses the following subset of the overall UPnP protocol stack. (The overall UPnP protocol stack is listed at the beginning of this document.)

A multi-homed device shall send description documents using the UPnP-enabled interface on which the HTTP GET request was received. To retrieve the UPnP description using a particular interface, a multi-homed control point shall use the URL provided in the discovery message which arrived on that interface.

**Figure 2-2: — Description retrieval protocol stack**



At the highest layer, description messages contain vendor-specific information, e.g., device type, service type, and required services. Moving down the stack, vendor content is supplemented by information from a UPnP Forum working committee, e.g., model name, model number, and specific URLs. Messages from the layers above are hosted in UPnP-specific protocols, defined in this document. In turn, the above messages are delivered via HTTP over TCP over IP. For reference, colors in [square brackets] above indicate which protocol defines specific header fields and body elements in the description messages listed below.

When a control point discovers a device on the network, it are allowed to wish to retrieve the Device Description document and Service Description Documents. Retrieving the UPnP device description is simple: the control point issues an HTTP GET request to the URL in the discovery message, and the device returns its description in the body of an HTTP response. Similarly, to retrieve a UPnP service description, the control point issues an HTTP GET request to the corresponding URL in the device description, and the device returns the description in the body of an HTTP response. The header fields and body for the response and request are explained in detail below.

First, a control point shall send a request with method GET in the following format. Values in *italics* are placeholders for actual values.

```
GET /descriptionPath HTTP/1.1
HOST: hostname:portNumber
ACCEPT-LANGUAGE: language preferred by control point
```

(No body for request to retrieve a description, but note that the message shall have a blank line following the last HTTP header field.)

Listed below are details for the request line and header fields appearing in the listing above. Field names are not case sensitive. All field values are case sensitive except where noted. See RFC 2616 and RFC 1945 for further requirements on encoding of values of these fields.

### Request line

GET

Method defined by HTTP. Can be GET or HEAD.

*descriptionPath*

Path component of device description URL (LOCATION header field in discovery message) or of the fully qualified service description URL. (If the SCPDURL sub element of the service element in the device description is an absolute URL, the fully qualified service description URL is the SCPDURL sub element. Otherwise (the SCPDURL sub element is a relative URL), the fully qualified service description URL is the URL resolved from the SCPDURL sub element in accordance with clause 5 of RFC 3986, using either the URLBase

element, if specified, or the URL from which the device description was retrieved as the base URL.) Single, absolute path (see also RFC 2616).

#### HTTP/1.1

The version supported by the control point. (Note: the control point shall implement all mandatory components of the version specified). are allowed to be any HTTP version that is backwards compatible to HTTP/1.0 (like HTTP/1.1).

#### Header fields

##### HOST

Required. Field value contains domain name or IP address and optional port components of device description URL (LOCATION header field in discovery message) or of the fully qualified service description URL. (If the SCPDURL sub element of the service element in the device description is an absolute URL, the fully qualified service description URL is the SCPDURL sub element. Otherwise (the SCPDURL sub element is a relative URL), the fully qualified service description URL is the URL resolved from the SCPDURL sub element in accordance with clause 5 of RFC 3986, using either the URLBase element, if specified, or the URL from which the device description was retrieved as the base URL.) If the port is empty or not given, port 80 is assumed.

##### ACCEPT-LANGUAGE

Allowed. Recommended for retrieving device descriptions. Field value contains preferred language(s) for description. If no description is available in this language, device is allowed to return a description in a default language. See RFC 1766 language tag(s).

After a control point sends a request, the device takes the second step and responds with a copy of its description. Including expected transmission time, a device shall respond within 30 seconds. If it fails to respond within this time, the control point should re-send the request. A device shall send a response in the following format and in accordance with clause 2.1, "Generic requirements on HTTP usage". Two example responses are provided below: one that uses the CONTENT-LENGTH header field, and one that uses chunked encoding (with 2 chunks). Values in *italics* are placeholders for actual values.

Note: XML namespace prefixes do not have to be the specific examples shown below (e.g., "s" or "u"); they can be any value that obeys the rules of the general XML namespace mechanism; a device shall accept action invocations that use other legal XML namespace prefixes.

#### Response using CONTENT-LENGTH header field – Valid with HTTP/1.0 or HTTP/1.1

```
HTTP/1.1 200 OK
CONTENT-LANGUAGE: language used in description
CONTENT-LENGTH: bytes in body
CONTENT-TYPE: text/xml; charset="utf-8"
DATE: when responded
```

*Body*

#### Response using chunked encoding – Valid with HTTP/1.1 only

```
HTTP/1.1 200 OK
TRANSFER-ENCODING: chunked
CONTENT-TYPE: text/xml; charset="utf-8"
CONTENT-LANGUAGE: language used in description
DATE: when responded
```

*Length of chunk 1 in hexadecimal notation*

*Chunk 1*

*Length of chunk 2 in hexadecimal notation*

*Chunk 2*

0

The body of this response is a UPnP device or service description as explained in detail above. Listed below are details for the header fields appearing in the listing above. Field names are not case sensitive. All field values are case sensitive except where noted.

#### Status Line

#### HTTP/1.1

The highest version supported by the origin server that is compatible with the control point that issued the request. For example, if the control point specified support for HTTP/1.0 in the request, the response shall contain HTTP/1.0.

#### 200 OK

HTTP defined status code indicating that no HTTP errors have occurred.

### Header fields

#### CONTENT-LANGUAGE

Required if and only if request included an ACCEPT-LANGUAGE header field. Field value contains language of description. RFC 1766 language tag(s).

#### CONTENT-LENGTH

Required if Origin Server does not close the session after sending the response AND Origin Server does not send the response using chunked encoding.

PROHIBITED if Origin Server sends the response using chunked encoding. Allowed otherwise.

Field value specifies the length of the body in bytes. Integer.

#### TRANSFER-ENCODING

Allowed for HTTP/1.1 and above. Field value specifies whether the response is chunked encoded by having field value "chunked". Shall NOT be specified if CONTENT-LENGTH header field is present.

#### CONTENT-TYPE

Required. Field value shall be "text/xml; charset="utf-8" " for description documents.

#### DATE

Recommended according to RFC 2616, clause 14.18. Field value contains date when response was generated. "rfc1123-date" as defined in RFC 2616.

#### SERVER

(No SERVER header field is required for description messages.)

Note that because HTTP 1.1 allows use of chunked encoding, some devices are allowed to send the description using chunked encoding if the GET request specifies HTTP 1.1. Therefore all implementations that include HTTP 1.1 client support shall support receiving chunked encoding.

### 2.12 References

ISO 8601, ISO (International Organization for Standardization). Representations of dates and times, 1988-06-15. Available at: <http://www.w3.org/TR/1998/NOTE-datetime-19980827>.

RFC 822, Standard for the format of ARPA Internet text messages. Available at: <http://www.ietf.org/rfc/rfc822.txt>.

RFC 1123, Includes format for dates, for, e.g., HTTP DATE header field. Available at: <http://www.ietf.org/rfc/rfc1123.txt>.

RFC 1766, Format for language tag for, e.g., HTTP ACCEPT-LANGUAGE header field. Available at: <http://www.ietf.org/rfc/rfc1766.txt>. See also <http://www.loc.gov/standards/iso639-2> for language codes.

RFC 2045, Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies. Available at: <http://www.ietf.org/rfc/rfc2045.txt>.

RFC 2046, Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types. Available at: <http://www.ietf.org/rfc/rfc2046.txt>.

RFC 2083, PNG (Portable Network Graphics) Specification Version 1.0. Available at: <http://www.ietf.org/rfc/rfc2083.txt>. See also <http://www.w3.org/TR/REC-png.html>.

RFC 2387, Format for representing content type, e.g., mimetype element for an icon. Available at: <http://www.ietf.org/rfc/rfc2387.txt>.

RFC 2616, HTTP: Hypertext Transfer Protocol 1.1. Available at: <http://www.ietf.org/rfc/rfc2616.txt>.

RFC 3986, Uniform Resource Identifiers (URI): Generic Syntax. Available at: <http://www.ietf.org/rfc/rfc3986.txt>.

UPC, Universal Product Code. 12-digit, all-numeric code that identifies the consumer package. Managed by the Uniform Code Council. Available at: [http://www.uc-council.org/main/ID\\_Numbers\\_and\\_Bar\\_Codes.html](http://www.uc-council.org/main/ID_Numbers_and_Bar_Codes.html).

XML, Extensible Markup Language. Available at: <http://www.w3.org/TR/2000/REC-xml-20001006>.

XML Schema (Part 1: Structures, Part 2: Datatypes). Available at: <http://www.w3.org/TR/xmlschema-1>, <http://www.w3.org/TR/xmlschema-2>.

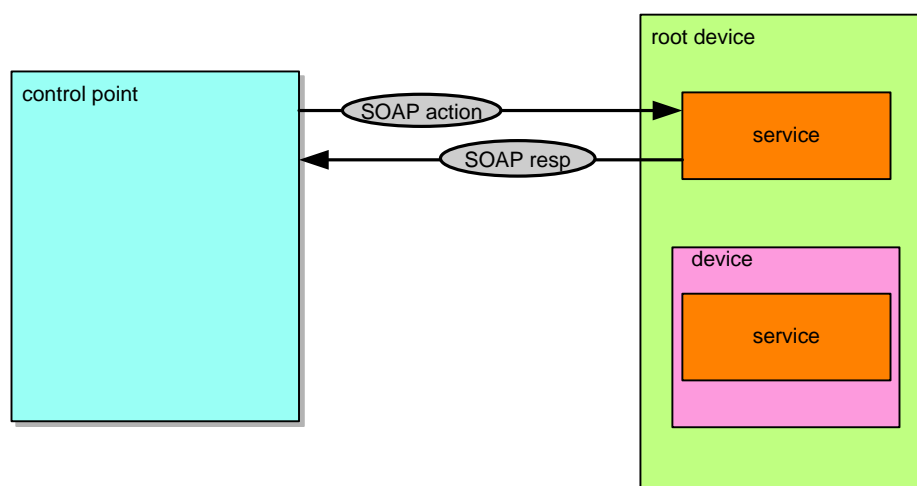
Namespaces in XML, Available at: <http://www.w3.org/TR/REC-xml-names/>.

### 3 Control

*Control is Step 3 in UPnP networking. Control comes after addressing (Step 0) where devices get a network address, after discovery (Step 1) where control points find interesting device(s), and after description (Step 2) where control points learn about device capabilities. Control is independent of eventing (Step 4) where control points listen to state changes in device(s). Through control, control points invoke actions on devices and poll for values. Control and eventing are complementary to presentation (Step 5) where control points display a user interface provided by device(s).*

Given knowledge of a device and its services, a control point can ask those services to invoke actions and receive responses indicating the result of the action. Invoking actions is a kind of remote procedure call; a control point sends the action to the device's service, and when the action has completed (or failed), the service returns any results or errors.

**Figure 3-1: — Control architecture**



To control a device, a control point invokes an action on the device's service. To do this, a control point sends a suitable control message to the fully qualified control URL for the service obtained from the controlURL sub element of the service element of the device description. If the controlURL sub element is an absolute URL, the fully qualified control URL for the service is the controlURL sub element. Otherwise (the controlURL sub element is a



relative URL), the fully qualified control URL for the service is the URL resolved from the controlURL sub element in accordance with clause 5 of RFC 3986, using either the URLBase element of the device description, if specified, or the URL from which the device description was retrieved as the base URL. A multi-homed control point that sends the control message on a particular interface shall use the fully qualified control URL from the description document received on that interface. In response, the service returns any results or errors from the action. The effects of the action, if any, is also allowed to be modeled by changes in the variables that describe the run-time state of the service. When these state variables change, events are published to all interested control points. This clause explains the protocol stack for, and format of, control messages. Clause 4, “Eventing” explains event publication.

Working committees and vendors are allowed to define actions to allow control points to determine the current value of one or more state variables. Similar to invoking an action, a control point sends the defined query message to the control URL for the service. In response, the service provides the value of the variable or variables; each service is responsible for keeping its state table consistent so control points can poll and receive meaningful values for those state variables for which query actions are defined. Clause 4, “Eventing” explains automatic notification of variable values.

As long as one of the discovery advertisements from a device have not expired, a control point is allowed to assume that the device and its services are available. If a device cancels at least one of its advertisements, a control point shall assume the device and its services are no longer available.

Control points and devices shall use UTF-8 for all UPnP control messages and responses.

While UDA does define a means to invoke actions and poll for values, UDA does not specify or constrain the design of an API for applications running on control points; OS vendors are allowed to create APIs that suit their customers’ needs.

If a large amount of data needs to be sent in association with an action (particularly if the amount of data is not known in advance), it is not recommended to send the data as part of a SOAP argument or as a MIME attachment to the SOAP message. Instead, it is recommended that out-of-band transfer be used. For example, a URL could be sent as an argument value, and an HTTP GET, PUT, or POST be used to transfer the data. HTTP chunked encoding can be used when the amount of data is not known in advance.

Responses to SOAP messages during the Control phase shall be sent to the same IP address from which the request was received. Any fully-qualified URLs contained in an action or response that refer to a resource on the device itself shall have the HOST portion of the URL set appropriately so that the resource will be reachable by the control point that requested the action. This might be accomplished by using the field value specified in the HTTP HOST header field of the control request.

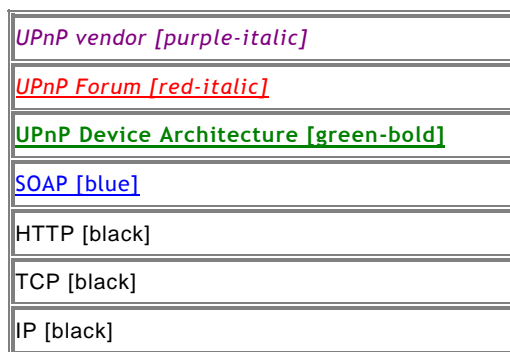
Services that use complex datatype arguments shall follow the requirements in clause 2.5, “Service description”

The remainder of this clause explains in detail how control messages are formatted and sent to devices.

### 3.1 Control protocols

To invoke actions and poll for values, control points (and devices) use the following subset of the overall UPnP protocol stack. (The overall UPnP protocol stack is listed at the beginning of this document.)

**Figure 3-2: — Control protocol stack**



At the highest layer, control messages contain vendor-specific information, e.g., argument values. Moving down the stack, vendor content is supplemented by information from a UPnP Forum working committee, e.g., action names, argument names, variable names. Messages from the layers above are hosted in UPnP-specific protocols, defined in this document. In turn, the above messages are formatted using a Simple Object Access Protocol (SOAP) header and body elements, and the messages are delivered via HTTP over TCP over IP. For reference, colors in [square brackets] above indicate which protocol defines specific header field elements in the subscription messages listed below.

The generic requirements on HTTP usage in UPnP 2.0 (as defined in clause 2.1, “Generic requirements on HTTP usage” of this document) shall be followed by devices and control points that implement Control.

#### 3.1.1 SOAP Profile

UPnP profiles SOAP 1.1, NOT REQUIRING that all devices support all allowed features of SOAP 1.1, but devices and control points shall support all mandatory features of SOAP 1.1. The following table summarizes the UPnP profiling of SOAP.

**Table 3-1: — SOAP 1.1 UPnP Profile**

<b>UPnP Control Request</b>	<b>Mandatory Optional Prohibited</b>	<b>Comment</b>
<Envelope> element	M	
encodingStyle attribute of <Envelope>	O	If present, shall be " <a href="http://schemas.xmlsoap.org/soap/encoding/">http://schemas.xmlsoap.org/soap/encoding/</a> "
<Header> element (child element of <Envelope>)	O	
actor attribute of <Header>	O	
mustUnderstand attribute of <Header>	O	Only allowed if defined by the service to which it is directed
encodingStyle attribute of <Header>	O	If present, shall be " <a href="http://schemas.xmlsoap.org/soap/encoding/">http://schemas.xmlsoap.org/soap/encoding/</a> "
<Body> element (child element of <Envelope>)	M	Exactly one <Body> child element allowed
encodingStyle attribute of <Body> element	P	
root Attribute of <Body> child element	O	Should not be used
<b>UPnP Control Response</b>		
<Envelope> element	M	
encodingStyle attribute of <Envelope>	O	If present, shall be " <a href="http://schemas.xmlsoap.org/soap/encoding/">http://schemas.xmlsoap.org/soap/encoding/</a> "
<Header> element (child element of <Envelope>)	O	
actor attribute of <Header>	O	
mustUnderstand attribute of <Header>	O	Only allowed if defined by the service to which it is directed
encodingStyle Attribute of <Header>	O	If present, shall be " <a href="http://schemas.xmlsoap.org/soap/encoding/">http://schemas.xmlsoap.org/soap/encoding/</a> "
<Body> element (child element of <Envelope>)	M	Exactly one <Body> child element allowed
encodingStyle attribute of <Body> element	P	
root attribute of <Body> child element	O	Should not be used
<b>UPnP Control Error Response</b>		
<Envelope> element	M	
encodingStyle attribute of <Envelope>	O	If present, shall be " <a href="http://schemas.xmlsoap.org/soap/encoding/">http://schemas.xmlsoap.org/soap/encoding/</a> "
<Body> element (child element of <Envelope>)	M	Exactly one <Body> child element allowed containing exactly one <Fault> child element
<Fault> child element of <Body>	M	
<faultcode> child element of <Fault>	M	
<faultstring> child element of <Fault>	M	
<detail> child element of <Fault>	M	

SOAP 1.1 allows the use of footers, which are disallowed in SOAP 1.2. A UPnP message shall not have any child elements of the <Envelope> element following the <Body> element.

## SOAP <Header> element

UPnP working committees and the UPnP technical committee is allowed to define allowed or mandatory SOAP header entries that are included in the SOAP <Header> element of UPnP action and UPnP action response messages. In addition, vendors are allowed to include other SOAP header entries in the SOAP <Header> element of UPnP action and UPnP action response messages. If there are no SOAP header entries in a message, the SOAP <Header> element can be omitted.

## SOAP [mustUnderstand](#) Attribute of <Header> element

The [mustUnderstand](#) attribute shall not be added to SOAP <Header> element targeted at (see also actor attribute below) standardized UPnP services or targeted at control points that interact with standardized UPnP services, unless its use has been explicitly defined by the UPnP technical committee or a working committee (e.g. UPnP security).

The [mustUnderstand](#) header attribute shall not be included on non-standard header entries that are targeted at (see also actor attribute) standardized services, as this breaks the basic interoperability of UPnP. [mustUnderstand](#) header entries are allowed to be included on non-standard header entries that are neither targeted at (see also actor attribute) standardized services (e.g. vendor defined services), nor targeted at control points interacting with standardized services.

**Table 3-2: — [mustUnderstand](#) attribute**

SOAP Node Type	v1.0	v1.1
Transmitting Node targeting a standardized service or a control point that interacts with a standardized service.	The <a href="#">mustUnderstand</a> attribute shall not be added to SOAP header entries, <i>unless</i> the UPnP technical committee or a working committee has explicitly defined its use.	
Transmitting Node targeting a vendor specific service or a vendor specific SOAP node.	Shall target endpoint (see actor clause below). The <a href="#">mustUnderstand</a> attribute is allowed to be used at the discretion of the vendor.	Is allowed to target intermediaries (see actor clause below). The <a href="#">mustUnderstand</a> attribute is allowed to be used at the discretion of the vendor.
Receiving Node	All unknown <Header> entries are ignored, except when explicitly defined differently by a working committee (UPnP Security).	All devices shall honor the actor (see actor clause below) and <a href="#">mustUnderstand</a> attributes. If a header entry with <a href="#">mustUnderstand</a> ="1" is not understood, the whole message fails and a <Faultcode> element shall be returned.

The SOAP [mustUnderstand](#) attribute has a restricted type of "xsd:boolean" that takes only "0" or "1" with "1" being true and "0" being false. A header entry with the [mustUnderstand](#) attribute set to a value of "1" shall be processed by targeted nodes or message processing shall fail. Such elements are considered "mandatory header entries". A SOAP node is considered to understand a SOAP header entry if that SOAP node understands the semantics specified for the XML expanded name of the outer-most element information item of that header entry. Mandatory SOAP header entries are presumed to modify the semantics of other SOAP header entries or SOAP <Body> elements and therefore shall be understood for correct semantics. UPnP nodes receiving header entries flagged with the [mustUnderstand](#) attribute shall process and understand mandatory header entries that are targeted at that node or the node shall NOT process the SOAP message at all. If a node fails to process or understand a mandatory entry, that node shall generate a SOAP [Fault](#) with the <faultcode> element set to "MustUnderstand". Support for mandatory header entries assures that key message parts that are targeted at a particular SOAP node will not be erroneously ignored.

If a `<Header>` entry is a mandatory `<Header>` entry and contains entries not understood by the targeted SOAP node, the SOAP node is allowed to attempt processing without understanding the semantics of the extensions. Mandatory extensions are not possible.

### SOAP actor Attribute of `<Header>` element

The SOAP [actor](#) attribute is used in SOAP 1.1 to identify the URI of SOAP node that is to process the `<Header>` entry. All SOAP nodes play the role of "http://schemas.xmlsoap.org/soap/actor/next", which is the first node (device or control point) that processes the message. The lack of an [actor](#) attribute indicates that the entry is targeted at the destination. All UPnP defined `<Header>` elements shall be targeted at the destination, unless explicitly defined otherwise by the UPnP technical committee or a working committee. Therefore, it is recommended that the actor attribute is not included on UPnP `<Header>` entries.

`<Header>` entries within messages that are sent to UPnP 1.0 devices or control points shall not be targeted at intermediaries (no actor attribute), since UPnP 1.0 devices and control points might ignore the actor attribute and parse a `<Header>` entry that is not intended for them.

If `<Header>` entries with an actor attribute are targeted at an intermediary and tagged [mustUnderstand="1"](#), the device or control point shall not return a SOAP Fault containing the `<faultcode>` element set to "MustUnderstand" due to failure to process the relevant `<Header>` element targeted at another entity.

### SOAP root Attribute

UPnP 2.0 REQUIRES that the first child element of the `<Body>` element shall be the root of the RPC request. Since UPnP 2.0 defines an RPC-architecture, there can only be one root. The serialization root should not use the [root](#) attribute, but it is NOT PROHIBITED.

### SOAP encodingStyle Attribute

UPnP 2.0 REQUIRES that devices and control points shall be able to receive messages that do not contain the SOAP [encodingStyle](#) attribute, as well as messages that contain the SOAP [encodingStyle](#) attribute with value "<http://schemas.xmlsoap.org/soap/encoding/>". When [encodingStyle](#) is not included, the [encodingStyle](#) is "http://schemas.xmlsoap.org/soap/encoding/".

When communicating with UPnP 1.0 devices or control points, an [encodingStyle](#) attribute shall be included on the SOAP `<Envelope>` element with value "<http://schemas.xmlsoap.org/soap/encoding/>". When communicating with UPnP 2.0 devices or control points, the [encodingStyle](#) attribute should be included and, if present, shall have the value "<http://schemas.xmlsoap.org/soap/encoding/>".

If additional encodings are needed for application data, applications are allowed to use out of band data encoding for the relevant data.

### SOAP `<Body>` element

UPnP 2.0 REQUIRES a `<Body>` element. It contains body entries for UPnP Actions and Responses. The actual entries are derived from the Service Description for the chosen Action. A response is either successful, in which case it contains output arguments, or unsuccessful, when it contains a `<Fault>` element as the only entry.

### SOAP `<Fault>` element of `<Body>` element

UPnP REQUIRES the use of SOAP `<Fault>` elements when a failure response is returned. Please see Table 3-2, "mustUnderstand attribute" on usage of the [mustUnderstand](#) attribute for how the `<detail>` element shall be constructed. When a `<Header>` element is encountered that is a mandatory `<Header>` element, the control point or device shall either

recognize the element or return the appropriate SOAP `<Fault>` element, containing the `<faultcode>` element set to “[MustUnderstand](#)”. Backwards-compatible services shall not use mandatory `<Header>` elements since previous UDAs allowed unknown `<Header>` elements to be ignored.

### Acceptable SOAP Character Encodings

All messages shall use UTF-8 serialization. The device or control point shall indicate the content type for all control messages using the HTTP “charset” parameter.

## 3.2 Actions

Control points are allowed to invoke actions on a device's services and receive results or errors back. The action, results, and errors are encapsulated in SOAP, sent via HTTP requests, and received via HTTP responses.

### 3.2.1 Action invocation

The Simple Object Access Protocol (SOAP) defines the use of XML and HTTP for remote procedure calls. UPnP 2.0 uses HTTP to deliver SOAP 1.1 encoded control messages to devices and return results or errors back to control points. See clause 2.1, “Generic requirements on HTTP usage” on use of HTTP in UPnP 2.0.

UPnP 2.0 deprecates the use of the HTTP Extension Framework (RFC 2774) for control. Specifically, control points shall send a request with method POST and shall not use the M-POST method. Devices shall not reject POST methods with a “405 Method Not Allowed” HTTP status code since this causes UPnP 1.0 control points to issue a request using M-POST.

Below is a listing of a control message sent using the POST method followed by an explanation of the header fields and body. To invoke an action on a device's service, a control point shall send a request with method POST in the following format. Two examples are provided: one using the CONTENT-LENGTH header and one using chunked encoding (with 2 chunks). Values in *italics* are placeholders for actual values.

Note: XML namespace prefixes do not have to be the specific examples shown below (e.g., “s” or “u”); they can be any value that obeys the rules of the general XML namespace mechanism; a device shall accept action invocations that use other legal XML namespace prefixes.

### Action invocation using the CONTENT-LENGTH header – Valid with HTTP/1.0 or HTTP/1.1

```
POST path control URL HTTP/1.0
HOST: hostname:portNumber
CONTENT-LENGTH: bytes in body
CONTENT-TYPE: text/xml; charset="utf-8"
USER-AGENT: OS/version UPnP/2.0 product/version
SOAPACTION: "urn:schemas-upnp-org:service:serviceType:v#actionName"

<?xml version="1.0"?>
<s:Envelope
  xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"
  s:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <s:Body>
    <u:actionName xmlns:u="urn:schemas-upnp-org:service:serviceType:v">
      <argumentName>in arg value</argumentName>
      <!-- other in args and their values go here, if any -->
    </u:actionName>
  </s:Body>
</s:Envelope>
```

## Action invocation using chunked encoding – Valid with HTTP/1.1 only

```
POST path control URL HTTP/1.1
HOST: hostname:portNumber
TRANSFER-ENCODING: "chunked"
CONTENT-TYPE: text/xml; charset="utf-8"
USER-AGENT: OS/version UPnP/2.0 product/version
SOAPACTION: "urn:schemas-upnp-org:service:serviceType:v#actionName"

Length of first chunk in hexadecimal notation

<?xml version="1.0"?>
<s:Envelope
  xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"
  s:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <s:Body>
    <u:actionName xmlns:u="urn:schemas-upnp-org:service:serviceType:v">
      <argumentName>in arg value</argumentName>
      <!-- other in args and their value go here, if any -->
    </u:actionName>
  </s:Body>

  Length of second chunk in hexadecimal notation

</s:Envelope>
0
```

Listed below are details for the request line, header fields, and body elements appearing in the listing above. Field names are not case sensitive. All HTTP field values and XML element names are case sensitive; XML values are not case sensitive except where noted. Except where noted, required elements shall occur exactly once (no duplicates), and recommended or allowed elements shall occur at most once.

### Request line

#### POST

Method defined by HTTP.

#### *path control URL*

Path component of the fully qualified control URL for this service. Single, absolute path (see also RFC 2616, clause 3.2.2).

#### HTTP/1.1

The version supported by the control point. (Note: the control point shall implement all mandatory components of the

version specified). Is allowed to be any HTTP version that is backwards compatible to HTTP/1.0 (like HTTP/1.1).

### Header fields

#### HOST

Required. Field value contains domain name or IP address and allowed port components of fully qualified control URL for this service. If the port is empty or not given, port 80 is assumed.

#### ACCEPT-LANGUAGE

PROHIBITED. The ACCEPT-LANGUAGE header field shall not be used in control messages.

#### CONTENT-LENGTH

Required if Origin Server does not close the session after sending the action invocation AND Origin Server does not send the action invocation using chunked encoding.

PROHIBITED if Origin Server sends the action invocation using chunked encoding. allowed otherwise.

Field value specifies the length of the body in bytes. Integer.

#### TRANSFER-ENCODING

Allowed for HTTP/1.1 and above. Field value specifies whether the action invocation is chunked encoded by having field value "chunked". shall not be specified if CONTENT-LENGTH header field is present.

#### CONTENT-TYPE

Required. Field value shall be "text/xml; charset="utf-8" ".

#### USER-AGENT

Allowed. Specified by UPnP vendor. String. Field value shall begin with the following "product tokens" (defined by HTTP/1.1). The first product token identifies the operating system in the form *OS name/OS version*, the second token represents the UPnP version and shall be [UPnP/2.0](#), and the third token identifies the product using the form *product name/product version*. For example, "USER-AGENT: *unix/5.1* [UPnP/2.0](#) *MyProduct/1.0*".

#### SOAPACTION

Required header field defined by SOAP. Field value shall be the service type, hash mark, and name of action to be invoked, all enclosed in double quotes. The specified service version shall indicate the version of the service that the control point wants to use while invoking the action. Its value may be any version of the service type in which the specified action was defined. When a control point invokes an action that has been defined in version K of a service, version number v shall be equal or higher than K. For example, if an action has been defined in version 2 of a service, it shall not be invoked using v=1. Furthermore; version v shall be a version that is supported by the device. For example, for devices that support only version 1 of a service, v shall be 1. Single URI.

### Body

<Envelope>

Required element defined by SOAP. `xmlns` namespace attribute shall be "<http://schemas.xmlsoap.org/soap/envelope/>". Shall include `encodingStyle` attribute with value "<http://schemas.xmlsoap.org/soap/encoding/>". A receiver shall generate a fault if it encounters a message whose <document> element has a local name of "Envelope" but a namespace name that is not "<http://schemas.xmlsoap.org/soap/envelope/>". Contains the following sub elements:

<Body>

Required element defined by SOAP. Shall be qualified with SOAP namespace. Contains the following entry:

<actionName>

Required. Name of element is name of action to invoke. `xmlns` namespace attribute shall be the service type enclosed in double quotes. The version specified shall be the same version specified in the SOAPACTION header field. Case sensitive. shall be the first child element of <Body>. Contains the following, ordered sub element(s):

<argumentName>

Required if and only if action has in arguments. Value to be passed to action. Repeat once for each in argument. (An element name is not qualified by a namespace; element nesting context is sufficient.) Case sensitive. Single data type as defined by UPnP service description. Every "in" argument in the definition of the action in the service description shall be included, in the same order as specified in the service description (SCPD) that is available from the device.

If the CONTENT-TYPE header field specifies an unsupported field value (other than "text/xml") the device shall return a "415 Unsupported Media Type" HTTP status code.

For future extensibility and according to the requirements in clause 2.7, "Non-standard vendor extensions" and clause 2.8, "UPnP Device Schema", when processing XML like the listing above, devices and control points shall ignore: (a) any unknown elements and their sub elements or content, and (b) any unknown attributes and their values.

Subject to the constraints defined in clause 2.7, "Non-standard vendor extensions" and clause 2.8, "UPnP Device Schema", control points and devices shall ignore any XML comments or XML processing instructions embedded in action requests that they do not understand.

When the value of any argument contains one or more characters reserved as markup (such as ampersand ("&") or less than ("<")), then the text shall be escaped in accordance with the provisions of clause 2.4 of the XML specification and each such character replaced with the



equivalent numeric representation or string (such as “&” or “<”). Such characters appearing in URLs are also allowed to be percent-encoded in accordance with the URL percent-encoding rules specified in clauses 2.1 and 2.4 of RFC 3986.

Note that because HTTP 1.1 allows use of chunked encoding, some control points are allowed to send the action request using chunked encoding if the POST method specifies HTTP 1.1. Device implementations that only support HTTP/1.0 and thus do not support receiving action requests using chunked encoding shall return a “505 HTTP Version Not Supported” HTTP status code. Control points shall not make HTTP 1.1 chunked POST requests to devices that are known to support only HTTP 1.0.

On a multi-homed control point, all fully qualified URLs contained in the action arguments that refer to resources on the control point shall be reachable on the interface on which the action request is sent.

### 3.2.2 Action Response

The service shall complete invoking the action and respond within 30 seconds, including expected transmission time (measured from the time the action message is transmitted until the time the associated response is received). Actions that take longer than this should be defined to return early and send an event when complete. If the service fails to respond within this time, what the control point should do is application-specific. A multi-homed device shall send the response on the same UPnP-enabled interface on which the request was received. The service shall send a successful completion response using the following format. The following two examples illustrate an action response using the CONTENT-LENGTH header and an action response using chunked encoding. The values in *italics* are placeholders for actual values.

Note; XML namespace prefixes do not have to be the specific examples shown below (e.g., “s” or “u”); they can be any value that obeys the rules of the general XML namespace mechanism; control points shall accept action responses that use other legal XML namespace prefixes.

#### Action response using the CONTENT-LENGTH header – Valid with HTTP/1.0 or HTTP/1.1

```
HTTP/1.0 200 OK
CONTENT-TYPE: text/xml; charset="utf-8"
DATE: when response was generated
SERVER: OS/version UPnP/2.0 product/version
CONTENT-LENGTH: bytes in body <?xml version="1.0"?>
<s:Envelope

  xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"

  s:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <s:Body>
    <u:actionNameResponse xmlns:u="urn:schemas-upnp-org:service:serviceType:v">
      <argumentName>out arg value</argumentName>
      <!-- other out args and their values go here, if any -->
    </u:actionNameResponse>
  </s:Body>
</s:Envelope>
```

## Action response using chunked encoding – Valid with HTTP/1.1 only

```
HTTP/1.1 200 OK
TRANSFER-ENCODING: "chunked"
CONTENT-TYPE: text/xml; charset="utf-8"
DATE: when response was generated
SERVER: OS/version UPnP/2.0 product/version

Length of first chunk in hexadecimal notation

<?xml version="1.0"?>
<s:Envelope
  xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"
  s:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <s:Body>
    <u:actionNameResponse xmlns:u="urn:schemas-upnp-org:service:serviceType:v">
      <argumentName>out arg value</argumentName>
      <!-- other out args and their values go here, if any -->
    </u:actionNameResponse>
  </s:Body>
</s:Envelope>
0
```

Listed below are details for the response line, header fields, and body elements appearing in the listing above. Field names are not case sensitive. All HTTP field values and XML element names are case sensitive; XML values are not case sensitive except where noted. Except where noted, required elements shall occur exactly once (no duplicates), and recommended or allowed elements are allowed to occur at most once.

### Response line

#### HTTP/1.1

The highest version supported by the origin server that is compatible with the control point that issued the request.

For example, if the control point specified support for HTTP/1.0 in the request, the response shall contain HTTP/1.0.

#### 200 OK

HTTP defined status code indicating that no HTTP errors were detected.

### Header fields

#### CONTENT-LANGUAGE

PROHIBITED. The CONTENT-LANGUAGE header field shall not be used in control messages.

#### CONTENT-LENGTH

Required if Origin Server does not close the session after sending the response AND Origin Server does not send the response using chunked encoding.

PROHIBITED if Origin Server sends the response using chunked encoding. Allowed otherwise.

Field value specifies the length of the body in bytes. Integer.

#### TRANSFER-ENCODING

Allowed for HTTP/1.1 and above. Field value specifies whether the response is chunked encoded by having field value "chunked" (in the example, the entire body is sent in a single chunk). Shall not be specified if CONTENT-LENGTH header field is present.

#### CONTENT-TYPE

Required. Field value shall be "text/xml; charset="utf-8" ".

#### DATE

Recommended. Field value contains date when response was generated. "rfc1123-date" as defined in RFC 2616.

## SERVER

Required. Specified by UPnP vendor. String. Field value shall begin with the following “product tokens” (defined by HTTP/1.1). The first product token identifies the operating system in the form *OS name/OS version*, the second token represents the UPnP version and shall be [UPnP/2.0](#), and the third token identifies the product using the form *product name/product version*. For example, “SERVER: *unix/5.1 UPnP/2.0 MyProduct/1.0*”.

## Body

<Envelope>

Required element defined by SOAP. `xmlns` namespace attribute shall be “<http://schemas.xmlsoap.org/soap/envelope/>”. shall include `encodingStyle` attribute with value “<http://schemas.xmlsoap.org/soap/encoding/>”. A receiver shall generate a fault if it encounters a message whose document element has a local name of “[Envelope](#)” but a namespace name that is not “<http://schemas.xmlsoap.org/soap/envelope/>”. Contains the following sub elements:

<Body>

Required element defined by SOAP. Shall be qualified with SOAP namespace. Contains the following entry:

<actionNameResponse>

Required. Name of element is action name prepended to [Response](#). `xmlns` namespace attribute shall be service type enclosed in double quotes. Devices that support the same action in multiple namespaces shall use the same namespace in the response as was used in the action invocation. For example, if an action was invoked using namespace:

`urn:schemas-upnp-org:service:ContentDirectory:2`

The response shall also use namespace:

`urn:schemas-upnp-org:service:ContentDirectory:2`

Case sensitive. shall be the first sub element of <Body>. Contains the following sub element:

<argumentName>

Required if and only if action has out arguments. Value returned from action. Repeat once for each out argument. If action has an argument marked with the `<retval/>` element, this argument shall be the first element. (An element name not qualified by a namespace; element nesting context is sufficient.) Case sensitive. Single data type as defined by UPnP service description. Every out argument in the definition of the action in the service description shall be included, in the same order as specified in the service description (SCPD) available from the device.

For future extensibility and according to the requirements in clause 2.7, “Non-standard vendor extensions” and clause 2.8, “UPnP Device Schema”, when processing XML like the listing above, devices and control points shall ignore: (a) any unknown elements and their sub elements or content, and (b) any unknown attributes and their values.

Subject to the constraints defined in clause 2.7, “Non-standard vendor extensions” and clause 2.8, “UPnP Device Schema”, control points and devices shall ignore any XML comments or XML processing instructions embedded in action responses that they do not understand.

On a multi-homed device, all fully qualified URLs contained in response arguments that refer to resources on the device shall be reachable on the UPnP-enabled interface on which the response message is sent.

### 3.2.3 UPnP Action Schema

The UPnP Action Schema defines the structures and data types used in the body of UPnP actions and action responses. As explained with the UPnP Device and Service Schemas, the UPnP Action Schema is written in XML syntax according to the conventions of XML Schema (Part 1: Structures, Part 2: Datatypes). The UPnP Action Schema is defined within a UPnP service template; however, the schema shall conform to the format as defined in clause B.3, “UPnP Control Schema”. The elements it defines are used in actions and action responses.

### 3.2.4 Recommendations and additional requirements

Control points and devices shall ignore any XML comments or XML processing instructions they may receive that they do not understand.

XML namespace prefixes do not have to be the specific examples given above (e.g., “s” or “u”); they can be any value that obeys the rules of the general XML namespace mechanism; control points shall accept responses that use other legal XML namespace prefixes.

If an action has no “out” arguments, it is valid to combine the opening and closing XML tags (e.g., “<actionNameResponse/>” instead of “<actionNameResponse></actionNameResponse>”).

When the value of any argument contains one or more characters reserved as markup (such as ampersand (“&”) or less than (“<”)), the text shall be escaped in accordance with the provisions of clause 2.4 of the XML specification and each such character replaced with the equivalent numeric representation or string (such as “&amp;” or “&lt;”). Such characters appearing in URLs are also allowed to be percent-encoded in accordance with the URL percent-encoding rules specified in clauses 2.1 and 2.4 of RFC 3986.

### 3.2.5 Action error response

Where the normal outcome of processing a SOAP message would have resulted in the transmission of a SOAP response, but rather a SOAP Fault is generated instead, a receiver shall transmit a SOAP Fault message in place of the response. If the service encounters an error while invoking the action sent by a control point, the service shall send a response within 30 seconds, including expected transmission time. Out arguments shall only be used to return data and shall not be used to convey error information. Error responses shall be sent using the following format. The following two examples illustrate an error response using the CONTENT-LENGTH header and an error response using chunked encoding. Values in *italics* are placeholders for actual values.

Note: XML namespace prefixes do not have to be the specific examples shown below (e.g., “s” or “u”); they can be any value that obeys the rules of the general XML namespace mechanism; control points shall error responses that use other legal XML namespace prefixes.

#### Error response using the CONTENT-LENGTH header – Valid using HTTP/1.0 and HTTP/1.1

```
HTTP/1.0 500 Internal Server Error
CONTENT-TYPE: text/xml; charset="utf-8"
DATE: when response was generated
SERVER: OS/version UPnP/2.0 product/version
CONTENT-LENGTH: bytes in body <?xml version="1.0"?>
<s:Envelope
  xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"
  s:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <s:Body>
    <s:Fault>
      <faultcode>s:Client</faultcode>
      <faultstring>UPnPError</faultstring>
      <detail>
        <UPnPError xmlns="urn:schemas-upnp-org:control-1-0">
          <errorCode>error code</errorCode>
          <errorDescription>error string</errorDescription>
        </UPnPError>
      </detail>
    </s:Fault>
  </s:Body>
</s:Envelope>
```

## Error response using chunked encoding – Valid using HTTP/1.1 only

```
HTTP/1.1 500 Internal Server Error
TRANSFER-ENCODING: "chunked"
CONTENT-TYPE: text/xml; charset="utf-8"
DATE: when response was generated
SERVER: OS/version UPnP/2.0 product/version

Length of first chunk in hexadecimal notation

<?xml version="1.0"?>
<s:Envelope
  xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"
  s:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <s:Body>
    <s:Fault>
      <faultcode>s:Client</faultcode>
      <faultstring>UPnPError</faultstring>
      <detail>
        <UPnPError xmlns="urn:schemas-upnp-org:control-1-0">
          <errorCode>error_code</errorCode>
          <errorDescription>error string</errorDescription>
        </UPnPError>
      </detail>
    </s:Fault>
  </s:Body>

Length of second chunk in hexadecimal notation

</s:Envelope>
0
```

Listed below are details for the response line, header fields, and body elements appearing in the listing above. HTTP field names are not case sensitive. All HTTP field values and XML element names are case sensitive; XML values are not case sensitive except where noted. Except where noted, required elements shall occur exactly once (no duplicates), and recommended or allowed elements are allowed to occur at most once.

### Response line

#### HTTP/1.1

The highest version supported by the origin server that is compatible with the control point that issued the request.

For example, if the control point specified support for HTTP/1.0 in the request, the response shall contain HTTP/1.0.

#### 500 Internal Server Error

HTTP defined status code indicating that an error has been detected.

### Header fields

#### CONTENT-LANGUAGE

PROHIBITED. The CONTENT-LANGUAGE header field shall not be used in control messages.

#### CONTENT-LENGTH

Required if Origin Server does not close the session after sending the response AND Origin Server does not send the response using chunked encoding.

PROHIBITED if Origin Server sends the response using chunked encoding. Allowed otherwise.

Field value specifies the length of the body in bytes. Integer.

#### TRANSFER-ENCODING

Allowed for HTTP/1.1 and above. Field value specifies whether the response is chunked encoded by having field value "chunked" (in the example above the body is sent in 2 chunks). shall not be specified if CONTENT-LENGTH header field is present.

CONTENT-TYPE

Required. Field value shall be "text/xml; charset="utf-8" ".

DATE

Recommended. Field value contains date when response was generated. "rfc1123-date" as defined in RFC 2616.

SERVER

Required. Specified by UPnP vendor. String. Field value shall begin with the following "product tokens" (defined by HTTP/1.1). The first product token identifies the operating system in the form *OS name/OS version*, the second token represents the UPnP version and shall be [UPnP/2.0](#), and the third token identifies the product using the form *product name/product version*. For example, "SERVER: *unix/5.1* [UPnP/2.0](#) *MyProduct/1.0*".

**Body**

<Envelope>

Required element defined by SOAP. `xmlns` namespace attribute shall be "<http://schemas.xmlsoap.org/soap/envelope/>". Shall include `encodingStyle` attribute with value "<http://schemas.xmlsoap.org/soap/encoding/>". A receiver shall generate a fault if it encounters a message whose document element has a local name of "Envelope" but a namespace name that is not "<http://schemas.xmlsoap.org/soap/envelope/>". Contains the following sub elements:

<Body>

Required element defined by SOAP. Shall be qualified with SOAP namespace. Contains the following sub element:

<Fault>

Required element defined by SOAP. Error encountered while invoking action. Shall be qualified with SOAP namespace. Contains the following sub elements:

<faultcode>

Required element defined by SOAP. Value shall be qualified with the SOAP namespace. Shall be "[Client](#)" for DCP specific errors. When mandatory header XML elements within the SOAP header cannot be processed it shall be the SOAP fault code "[MustUnderstand](#)".

<faultstring>

Required element defined by SOAP. Shall be "[UPnPError](#)" for DCP specific errors.

<detail>

Required element defined by SOAP. Contains the following subelement:

<UPnPError>

Required element for DCP specific errors. Is allowed to be empty for other errors. Contains the following subelements:

<errorCode>

Required element defined by UDA. Code identifying what error was encountered. See Table 3-3, "UPnP Defined Action error codes" for values. Integer.

<errorDescription>

Recommended element defined by UDA. Short description. See Table 3-3, "UPnP Defined Action error codes" for recommended values; other values Is allowed to be used by vendors. Human-readable string. Recommended < 256 characters.

The following table summarizes defined error types and the corresponding value for the <errorCode> and <errorDescription> elements.

**Table 3-3: — UPnP Defined Action error codes**

ErrorCode	errorDescription	Description
-----------	------------------	-------------

ErrorCode	errorDescription	Description
401	Invalid Action	No action by that name at this service.
402	Invalid Args	Could be any of the following: not enough in args, args in the wrong order, one or more in args are of the wrong data type. Additionally, the UPnP Certification Test Tool shall return the following warning message if there are too many in args: 'Sending too many in args is not recommended and may cause unexpected results.'
403	<i>(Do Not Use)</i>	<i>(This code has been deprecated.)</i>
501	Action Failed	Is allowed to be returned if current state of service prevents invoking that action.
600	Argument Value Invalid	The argument value is invalid
601	Argument Value Out of Range	An argument value is less than the <b>minimum</b> or more than the <b>maximum</b> value of the allowed value range, or is not in the allowed value list.
602	Optional Action Not Implemented	The requested action is optional and is not implemented by the device.
603	Out of Memory	The device does not have sufficient memory available to complete the action. This is allowed to be a temporary condition; the control point is allowed to choose to retry the unmodified request again later and it is expected to succeed if memory is available.
604	Human Intervention Required	The device has encountered an error condition which it cannot resolve itself and required human intervention such as a reset or power cycle. See the device display or documentation for further guidance.
605	String Argument Too Long	A string argument is too long for the device to handle properly.
606-612 <sup>4</sup>	<i>Reserved</i>	These ErrorCodes are reserved for UPnP DeviceSecurity.
613-699	<i>TBD</i>	Common action errors. Defined by UPnP Forum Technical Committee.
700-799	<i>TBD</i>	Action-specific errors defined by UPnP Forum working committee.
800-899	<i>TBD</i>	Action-specific errors for non-standard actions. Defined by UPnP vendor.

### 3.2.6 UPnP Error Schema

The UPnP Error Schema defines the structures and data types used in the body of UPnP error messages. As with the UPnP Device and Service Schemas, the UPnP Error Schema is written in XML syntax and according to the conventions of XML Schema (Part 1: Structures, Part 2: Datatypes). clause B.4, “UPnP Error Schema” contains a listing of this schema. The elements it defines are used in error messages.

For future extensibility and according to the requirements in clause 2.7, “Non-standard vendor extensions” and clause 2.8, “UPnP Device Schema”, when processing XML like the listing above, devices and control points shall ignore: (a) any unknown elements and their sub elements or content, and (b) any unknown attributes and their values.

Subject to the constraints defined in clause 2.7, “Non-standard vendor extensions” and clause 2.8, “UPnP Device Schema”, control points and devices shall ignore any XML comments or XML processing instructions embedded in UPnP device and service descriptions that they do not understand.

XML namespace prefixes do not have to be the specific examples given above (e.g., “s” or “u”); they can be any value that obeys the rules of the general XML namespace mechanism; control points shall accept responses that use other legal XML namespace prefixes.

### 3.3 Query for variable

*The QueryStateVariable action has been deprecated by the UPnP Forum and shall not be used by control points except in limited testing scenarios. Working committees and vendors shall explicitly define actions for querying of state variables for which this capability is desired. Such explicit query actions is allowed to include multiple state variables, if desired. For the full definition of QueryStateVariable see the UPnP 1.0 specification.*

### 3.4 References

RFC 1123, Includes format for dates, for, e.g., HTTP DATE header field. Available at: <http://www.ietf.org/rfc/rfc1123.txt>.

RFC 2616, HTTP: Hypertext Transfer Protocol 1.1. Available at: <http://www.ietf.org/rfc/rfc2616.txt>.

RFC 2774, HTTP Extension Framework. Available at: <http://www.ietf.org/rfc/rfc2774.txt>.

RFC 3986, Uniform Resource Identifiers (URI): Generic Syntax. Available at: <http://www.ietf.org/rfc/rfc3986.txt>.

SOAP, Simple Object Access Protocol. Available at: <http://www.w3.org/TR/2000/NOTE-SOAP-20000508>.

XML, Extensible Markup Language. Available at: <http://www.w3.org/XML>.

XML Schema (Part 1: Structures, Part 2: Datatypes), Available at: <http://www.w3.org/TR/xmlschema-1>, <http://www.w3.org/TR/xmlschema-2>.

## 4 Eventing

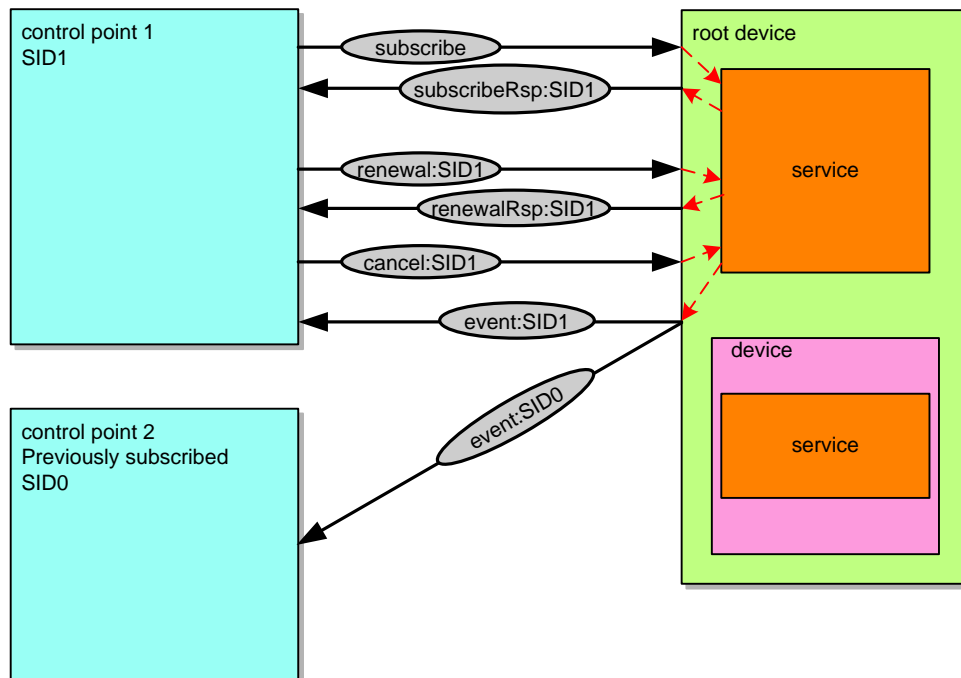
*Eventing is Step 4 in UPnP networking. Eventing comes after addressing (Step 0) where devices get a network address, after discovery (Step 1) where control points find interesting device(s), and after description (Step 2) where control points learn about device capabilities. Eventing is intimately linked with control (Step 3) where control points send actions to devices. Through eventing, control points listen to state changes in device(s). Control and eventing are complementary to presentation (Step 5) where control points display a user interface provided by device(s).*

After a control point has (1) discovered a device and (2) retrieved a description of the device and its services, the control point has the essentials for eventing. As clause 2, “Description” explains, a UPnP service description includes a list of actions the service responds to and a list of variables that model the state of the service at run time. If one or more of these state variables are evented, then the service publishes updates when these variables change, and a control point is allowed to subscribe to receive this information. Two types of eventing are supported by this specification: unicast eventing as found in version 1.0 of the UPnP specification where a control point is allowed to subscribe to receive variable updates; and multicast eventing where variables is allowed to be defined as multicast events and can be additionally sent over UDP to any listening device on the multicast event address. This form of eventing is useful when events which are not relevant to a specific UPnP interaction should be delivered to *control points* to inform users, and when multiple *controlled devices* want to inform multiple other *controlled devices*. Throughout this clause, *publisher* refers to the source of the events (typically a device's service), *subscriber* refers to the destination of events (typically a control point), and the term *receiver* refers to the listener of multicast events (typically a control point, but is allowed to also be a controlled device).



## 4.1 Unicast eventing

Figure 4-1: — Unicast eventing architecture



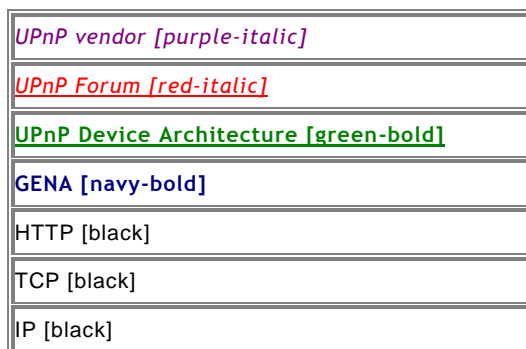
To subscribe to eventing, a subscriber sends a *subscription message*. If the subscription is accepted, the publisher responds with a duration for the subscription. To keep the subscription active, a subscriber shall renew its subscription before the subscription expires. When a subscriber no longer needs eventing from a publisher, the subscriber should cancel its subscription. This clause explains subscription, renewal, and cancellation messages in detail below.

The publisher notes changes to state variables by sending *event messages*. Event messages contain the names of one or more state variables and the current value of those variables, expressed in XML. A special *initial event message* is sent when a subscriber first subscribes; this event message contains the names and values for all evented variables and allows the subscriber to initialize its model of the state of the service. To support scenarios with multiple control points, eventing can be used to keep interested control points informed about the effects of actions performed by other control points or using other mechanisms for device control (such as front panel controls). All subscribers are sent all event messages, subscribers receive event messages for all evented variables (not just some), and event messages are sent no matter why the state variable changed (either in response to a requested action or because the state the service is modeling changed). This clause explains the format of event messages in detail below.

Some state variables are allowed to change value too rapidly for eventing to be useful. One alternative is to filter, or moderate, the number of event messages sent due to changes in a variable's value. Some state variables are allowed to contain values too large for eventing to be useful; for this, or other reasons, a service is allowed to designate one or more state variables as *non evented* and never send event messages to subscribers. To determine the current value for such non-evented variables, control points shall poll the service explicitly, presuming that an action is provided to obtain the value of the state variable. This clause explains how variable eventing is described within a service description.

To send and receive subscription and event messages, control points and services use the following subset of the overall UPnP protocol stack. (The overall UPnP protocol stack is listed at the beginning of this document.)

**Figure 4-2: — Unicast eventing protocol stack**



At the highest layer, subscription and event messages contain vendor-specific information like URLs for subscription and duration of subscriptions or specific variable values. Moving down the stack, vendor content is supplemented by information from a UPnP Forum working committee, like service identifiers or variable names. Messages from the layers above are hosted in UPnP-specific protocols, defined in this document. In turn, the above messages are delivered via HTTP that has been extended using additional methods and header fields. The HTTP messages are delivered via TCP over IP. For reference, colors in [square brackets] above indicate which protocol defines specific header fields in the subscription messages listed below.

The remainder of this clause first explains subscription, including details of subscription messages, renewal messages, and cancellation messages. Second, it explains in detail how event messages are formatted and sent to control points, and the initial event message. Finally, it explains the UPnP Device and Service Schemas as they pertain to eventing.

The generic requirements on HTTP usage in UPnP 2.0 (as defined in clause 2.1, “Generic requirements on HTTP usage” of this document) shall be followed by devices and control points that implement eventing.

Services that use evented complex datatypes shall follow the requirements in clause 2.5, “Service description”.

#### 4.1.1 Subscription

A service has eventing if and only if one or more of the state variables are evented.

If a service has eventing, it publishes event messages to interested subscribers. The publisher maintains a list of subscribers, keeping for each subscriber the following information.

unique subscription identifier

Required. Shall be unique over the lifetime of the subscription, however long or short that may be. Generated by publisher in response to subscription message. RECOMMEND universally-unique identifiers to ensure uniqueness. Single URI.

delivery URL for event messages

Required. Provided by subscriber in subscription message. Single URL.

event key

Required. Key is 0 for initial event message. Key shall be sequentially numbered for each subsequent event message; subscribers can verify that no event messages have been lost if the subscriber has received sequentially numbered event keys. Shall wrap from 4294967295 to 1 (32-bit unsigned decimal integer). Implementations are allowed to include leading “0” characters in the event key, which shall be ignored.

subscription duration

Required. Amount of time, or duration until subscription expires. Single integer, preceded in subscription messages by the keyword “**Second-**” (no spaces). UPnP 1.0 defines the use of the keyword **infinite** instead of an integer. This keyword is deprecated in UPnP 2.0 (it leads to problems if control points disappear without unsubscribing and is hardly used): UPnP 2.0 control points shall not subscribe using keyword **infinite**, UPnP 2.0 devices shall not set actual subscription durations to “infinite”. The presence of **infinite** in a request shall

be silently ignored by a UPnP 2.0 device (the presence of infinite is handled by the device as if the TIMEOUT header field in a request was not present) . The keyword infinite shall not be returned by a UPnP 2.0 device.

HTTP version supported by the subscriber

Required if the publisher supports chunked encoding of event notification messages, so that chunked messages are *not* sent to subscribers that do not support them.

A multi-homed publisher shall also maintain information on the UPnP-enabled interface on which each subscription message was received. The same interface shall be used when publishing event messages to the corresponding subscriber.

The publisher should accept as many subscriptions as it can reasonably maintain, taking into account that the number of event messages that need to be delivered per event, which increases linearly with the number of subscriptions.

The list of subscribers is updated via subscription, renewal, and cancellation messages explained immediately below and event messages explained later in this clause.

To subscribe to eventing for a service, a subscriber sends a *subscription message* containing a URL for the publisher, a service identifier for the publisher, and a delivery URL for event messages. The subscription message MAY also include a requested duration for the subscription. The URL and service identifier for the publisher come from a description message. As clause 2, “Description” explains, a description message contains a device description. A device description contains (among other things), for each service, an eventing URL (obtained from the eventSubURL element) and a service identifier (in the serviceId element); these correspond to the URL and service identifier for the publisher, respectively. If eventSubURL is an absolute URL, the fully qualified event subscription URL is the eventSubURL. If eventSubURL is a relative URL, the fully qualified event subscription URL is the URL resolved from eventSubURL in accordance with clause 5 of RFC 3986, using either the URLBase element, if specified, or the URL from which the device description was retrieved as the base URL. If the eventSubURL is empty, no subscriptions are possible. The fully qualified event subscription URL for the publisher shall be unique to a particular service within this device. A multi-homed control point that sends the subscription message on a particular UPnP-enabled interface shall use the fully qualified eventing URL from the description document received on that UPnP-enabled interface. The delivery URL contained in the subscription message shall be reachable on that interface.

The subscription message is a request to receive all event messages. STATEVAR is a recommended mechanism to subscribe to event messages on a variable-by-variable basis. STATEVAR allows a subscriber to avoid being sent all event messages from the service. Using STATEVAR, or not, is to be considered when designing a service. Especially when you are using a large number of variables, STATEVAR will allow you to eliminate unnecessary traffic.

If the subscription is accepted, the publisher responds with a unique identifier for this subscription and a duration for this subscription. A duration should be chosen that matches assumptions about how frequently control points are removed from the network; if control points are removed every few minutes, then the duration should be similarly short, allowing a publisher to rapidly deprecate any expired subscribers; if control points are expected to be semi-permanent, then the duration should be very long, minimizing the processing and traffic associated with renewing subscriptions.

As soon as possible after the subscription is accepted, the publisher also sends the first, or *initial* event message to the subscriber. This message includes the names and current values for all evented variables. (The data type and range for each variable is described in a service description. Clause 2, “Description” explains this in more detail.) This initial event message is always sent, even if the control point unsubscribes before it is delivered. The device shall insure that the control point has received the response to the subscription request before sending the initial event message, to insure that the control point has received the SID (subscription ID) and can thereby correlate the event message to the subscription.

To keep the subscription active, a subscriber shall renew its subscription before the subscription expires by sending a renewal message. The renewal message is sent to the same URL as the subscription message, but the renewal message does not include a delivery URL for event messages; instead the renewal message includes the subscription identifier. The response for a renewal message is the same as one for a subscription message.

If a subscription expires, the subscription identifier becomes invalid, and the publisher stops sending event messages to the subscriber and can clean up its list of subscribers. If the subscriber tries to send any message other than a subscription message, the publisher shall reject the message because the subscription identifier is invalid.

When a subscriber no longer needs eventing from a particular service, the subscriber should cancel its subscription. Canceling a subscription generally reduces service, control point, and network load. If a subscriber is removed abruptly from the network, it might be impossible to send a cancellation message. As a fallback, the subscription will eventually expire on its own unless renewed.

It is strongly recommended that subscribers monitor discovery messages from the publisher. If the publisher cancels its advertisements or if the value of the BOOTID.UPNP.ORG is increased without a prior **ssdp:update** message with a matching NEXTBOOTID.UPNP.ORG field value, subscribers shall assume that their subscriptions have been cancelled.

Below is an explanation of the specific format of requests, responses, and errors for subscription, renewal, and cancellation messages.

#### 4.1.2 SUBSCRIBE with NT and CALLBACK

For each service in a device, a description message contains an event subscription URL (obtained from the eventSubURL sub element of service element in the device description) and the UPnP service identifier (serviceld sub element in service element in device description). To subscribe to eventing for a particular service, a subscription message is sent to that service's fully qualified event subscription URL. If eventSubURL is an absolute URL, the fully qualified event subscription URL is the eventSubURL. Otherwise, if eventSubURL is a relative URL, the fully qualified event subscription URL is the URL resolved from eventSubURL in accordance with clause 5 of RFC 3986, using either the URLBase element, if specified, or the URL from which the device description was retrieved as the base URL. The message contains that service's identifier as well as a delivery URL for event messages. A multi-homed control point that sends the subscription message on a particular UPnP-enabled interface shall use the fully qualified eventing URL from the description document received on that interface. The delivery URL contained in the subscription message shall be reachable on that interface. A subscription message MAY also include a requested subscription duration.

To subscribe to eventing for a service, a subscriber shall send a request with method SUBSCRIBE and NT and CALLBACK header fields in the following format. Values in *italics* are placeholders for actual values.

```
SUBSCRIBE publisher path HTTP/1.1
HOST: publisher host:publisher port
USER-AGENT: OS/version UPnP/2.0 product/version
CALLBACK: <delivery URL>
NT: upnp:event
TIMEOUT: Second-requested subscription duration
STATEVAR: CSV of Statevariables
```

(No body for request with method SUBSCRIBE, but note that the message shall have a blank line following the last HTTP header field.)

Listed below are details for the request line and header fields appearing in the listing above. Field names are not case sensitive. All field values are case sensitive except where noted.

## Request line

### SUBSCRIBE

Method to initiate or renew a subscription.

### publisher path

Path component of the fully qualified event subscription URL. Single, absolute path (see also RFC 2616, clause 3.2.2).

### HTTP/1.1

The version supported by the control point. (Note: the control point shall implement all mandatory components of the

version specified). MAY be any HTTP version that is backwards compatible to HTTP/1.0 (like HTTP/1.1).

## Header fields

### HOST

Required. Field value contains domain name or IP address and optional port components of the fully qualified event subscription URL. If the port is missing or empty, port 80 is assumed.

### USER-AGENT

Allowed. Specified by UPnP vendor. String. Field value shall begin with the following “product tokens” (defined by HTTP/1.1). The first product token identifies the operating system in the form *OS name/OS version*, the second token represents the UPnP version and shall be UPnP/2.0, and the third token identifies the product using the form *product name/product version*. For example, “USER-AGENT: *unix/5.1* UPnP/2.0 *MyProduct/1.0*”. CALLBACK

Required. Field value contains location to send event messages to. Defined by UPnP vendor. If there is more than one URL, when the service sends events, it will try these URLs in order until one succeeds. One or more URLs each enclosed by angle brackets (“<” and “>”). Each URL shall be an HTTP over TCP URL (prefixed by “http://”). The device shall not truncate this URL in any way; if insufficient memory is available to store the entire CALLBACK URL, the device shall reject the subscription. At least one of the delivery URLs shall be reachable by the device.

### NT

Required. Field value contains Notification Type. shall be upnp:event.

### SID

(No SID header field is used to subscribe.)

### TIMEOUT

Recommended. Field value contains requested duration until subscription expires. Consists of the keyword **Second-** followed (without an intervening space) by an integer. UPnP 1.0 defined that the integer can be replaced by the keyword **infinite**. This has been deprecated in UPnP 2.0: UPnP 2.0 control points shall not subscribe using keyword **infinite**.

### STATEVAR

Recommended. Field value contains requested list of state variables. Consists of an CSV list of evented state variables that the control point wants to subscribe to. The device implementation will acknowledge the subscribed state variables in the subscription response. Note that when the device implementation does not recognize this field, the acknowledgement of the registered state variables will not be sent, and the events generated by the subscription will contain all implemented evented state variables in the service.

If there are enough resources to maintain the subscription, the publisher should accept it. To accept a subscription request, a publisher shall send a response in the following format within 30 seconds, including expected transmission time. This shall be sent to the same endpoint as that over which the subscription request was sent. After accepting the subscription, the publisher assigns a unique identifier for the subscription, assigns a duration for the subscription, and sends an initial event message (explained in detail later in this clause). A multi-homed publisher shall send the response on the same UPnP-enabled interface on which the subscription message was received. Values in *italics* are placeholders for actual values.

```
HTTP/1.1 200 OK
DATE: when response was generated
SERVER: OS/version UPnP/2.0 product/version
SID: uuid:subscription-UUID
CONTENT-LENGTH: 0
TIMEOUT: Second-actual subscription duration
ACCEPTED-STATEVAR: CSV of state variables
```

(No body for response to a request with method SUBSCRIBE, but note that the message shall have a blank line following the last HTTP header field.)

If the device sends the response over HTTP/1.0 without setting the KeepAlive token, or over HTTP/1.1 with the CONNECTION: close header field, the device shall insure that the TCP FIN flag is sent BEFORE sending the initial event message. In all other cases, (unless the response is chunked), a CONTENT-LENGTH shall be specified, (and set to 0), prior to sending the initial event.

Listed below are details for header fields appearing in the listing above. Field names are not case sensitive. All field values are case sensitive except where noted.

## Response line

### HTTP/1.1

The highest version supported by the origin server that is compatible with the control point that issued the request.

For example, if the control point specified support for HTTP/1.0 in the request, the response shall contain HTTP/1.0.

### 200 OK

HTTP defined status code indicating that no HTTP errors were detected..

## Header fields

### DATE

Recommended. Field value contains date when the response was generated. "rfc1123-date" as defined in RFC 2616.

### SERVER

Required. Specified by UPnP vendor. String. Field value shall begin with the following "product tokens" (defined by HTTP/1.1). The first product token identifies the operating system in the form *OS name/OS version*, the second token represents the UPnP version and shall be *UPnP/2.0*, and the third token identifies the product using the form *product name/product version*. For example, "SERVER: *unix/5.1 UPnP/2.0 MyProduct/1.0*". SID

Required. Field value contains Subscription Identifier. Shall be universally unique. Shall begin with uuid:. Defined by UPnP vendor. See clause 1.1.4, "UUID format and recommended generation algorithms" for the mandatory UUID format.

### TIMEOUT

Required. Field value contains actual duration until subscription expires. Keyword "**Second-**" followed by an integer (no space). Should be greater than or equal to 1800 seconds (30 minutes).

### CONTENT-LENGTH

Required if TCP FIN flag cannot be guaranteed to be sent before the initial event is sent. shall have field value "0".

### ACCEPTED-STATEVAR

Required if the SUBSCRIPTION contains the STATEVAR header containing a valid CSV of implemented evented state variables. The implementation then shall only send events for the state variables listed in the STATEVAR list. When one of the listed state variables in the STATEVAR CSV list is not implemented, the ACCEPTED\_STATEVAR header shall not be sent back. This means that the subscription is valid and all state variables designated to be evented shall be evented.

If a publisher cannot accept the subscription, or if there is an error with the subscription request, the publisher shall send a response with one of the following errors. The response shall be sent within 30 seconds, including expected transmission time.

**Table 4-4: — HTTP Status Codes indicating a Subscription Error**

ErrorCode	errorDescription	Description
400	Incompatible header fields	An SID header field and one of NT or CALLBACK header fields are present.
412	Precondition Failed	CALLBACK header field is missing or does not contain a valid HTTP URL; or the NT header field does not equal <a href="#">upnp:event</a> .
5xx	Unable to accept renewal	If the publisher is unable to accept a renewal, it shall respond with an appropriate 500-series HTTP status code.

Other errors, including other HTTP status codes, MAY be returned by layers in the protocol stack below the UPnP protocols. Consult documentation on those protocols for details.

### 4.1.3 Renewing a subscription with SUBSCRIBE with SID

To renew a subscription to eventing for a particular service, a renewal message is sent to that service's fully qualified event subscription URL (See clause 4.1.2, "SUBSCRIBE with NT and CALLBACK"). However, unlike an initial subscription message, a renewal message does not contain either the service's identifier nor a delivery URL for event messages. Instead, the message contains the *subscription* identifier assigned by the publisher, providing an unambiguous reference to the subscription to be renewed. Like a subscription message, a renewal message MAY also include a requested subscription duration. A multi-homed control point shall send the renewal message using the same pair of UPnP-enabled interfaces used for the initial subscription.

The renewal message uses the same method as the subscription message, but the two messages use a disjoint set of header fields; renewal uses SID and subscription uses NT and CALLBACK. A message that includes SID and either of NT or CALLBACK header fields is an error.

To renew a subscription to eventing for a service, a subscriber shall send a request with method SUBSCRIBE and SID header field in the following format. Values in *italics* are placeholders for actual values.

```
SUBSCRIBE publisher path HTTP/1.1
HOST: publisher host:publisher port
SID: uuid:subscription UUID
TIMEOUT: Second-requested subscription duration
```

(No body for method with request SUBSCRIBE, but note that the message shall have a blank line following the last HTTP header field.)

Listed below are details for the request line and header fields appearing in the listing above. Field names are not case sensitive. All field values are case sensitive except where noted.

#### Request line

SUBSCRIBE

Method to initiate or renew a subscription.

*publisher path*

Path component of the fully qualified event subscription URL. Single, absolute path (see also RFC 2616, clause 3.2.2).

HTTP/1.1

The version supported by the control point. (Note: the control point shall implement all mandatory components of the version specified). MAY be any HTTP version that is backwards compatible to HTTP/1.0 (like HTTP/1.1)

**Header fields**

HOST

Required. Field value contains domain name or IP address and optional port components of fully qualified event subscription URL. If the port is missing or empty, port 80 is assumed.

CALLBACK

(No CALLBACK header field is used to renew an event subscription.)

NT

(No NT header field is used to renew an event subscription.)

SID

Required. Field value contains Subscription Identifier. Shall be the subscription identifier assigned by publisher in response to subscription request. Shall be universally unique. Shall begin with uuid:. Defined by UPnP vendor. See clause 1.1.4, "UUID format and recommended generation algorithms for the mandatory UUID format.

TIMEOUT

Recommended. Field value contains requested duration until subscription expires. Keyword **Second-** followed by an integer (no space). UPnP 1.0 defined that the integer can be replaced by the keyword **infinite**. This has been deprecated in UPnP 2.0: UPnP 2.0 control points shall not subscribe using keyword **infinite**. See reference above.

To accept a renewal, the publisher reassigns a duration for the subscription and shall send a response in the same format and with the same conditions as a response to a request for a new subscription, except that the initial event message is not sent again.

If a publisher cannot accept the renewal, or if there is an error with the renewal request, the publisher shall send a response with one of the following errors. The response shall be sent within 30 seconds, including expected transmission time.

**Table 4-5: — HTTP Status Codes indicating a Resubscription Error**

ErrorCode	errorDescription	Description
400	Incompatible header fields	An SID header field and one of NT or CALLBACK header fields are present.
412	Precondition Failed	An SID does not correspond to a known, un-expired subscription; or the SID header field is missing or empty.
5xx	Unable to accept renewal	If the publisher is unable to accept a renewal, it shall respond with an appropriate 500-series HTTP status code.

Other errors, including other HTTP status codes, MAY be returned by layers in the protocol stack below the UPnP protocols. Consult documentation on those protocols for details.

**4.1.4 Canceling a subscription with UNSUBSCRIBE**

When eventing is no longer needed from a particular service, a cancellation message should be sent to that service's fully qualified event subscription URL (see clause 4.1.2, "SUBSCRIBE with NT and CALLBACK"). The message contains the subscription identifier. A multi-homed control point shall send the cancellation message using the same pair of UPnP-enabled interfaces used for the initial subscription. Canceling a subscription generally reduces service, control point, and network load. If a control point is removed abruptly from the network, it might be impossible to send a cancellation message. As a fallback, the subscription will eventually expire on its own unless renewed.



To explicitly cancel a subscription to eventing for a service, a subscriber shall send a request with method UNSUBSCRIBE in the following format. Values in *italics* are placeholders for actual values.

```
UNSUBSCRIBE publisher path HTTP/1.1
HOST: publisher host:publisher port
SID: uuid:subscription UUID
```

(No body for request with method UNSUBSCRIBE, but note that the message shall have a blank line following the last HTTP header field.)

Listed below are details for the request line and header fields appearing in the listing above. Field names are not case sensitive. All field values are case sensitive except where noted.

### Request line

#### UNSUBSCRIBE

Method to cancel a subscription.

#### *publisher path*

Path component of the fully qualified event subscription URL. Single, absolute path (see also RFC 2616, clause 3.2.2).

#### HTTP/1.1

The version supported by the control point. (Note: the control point shall implement all mandatory components of the

version specified). MAY be any HTTP version that is backwards compatible to HTTP/1.0 (like HTTP/1.1)

### Header fields

#### HOST

Required. Field value contains domain name or IP address and optional port components of fully qualified event subscription URL. If the port is missing or empty, port 80 is assumed.

#### CALLBACK

(No CALLBACK header field is used to cancel an event subscription.)

#### NT

(No NT header field is used to cancel an event subscription.)

#### SID

Required. Field value contains Subscription Identifier. Shall be the subscription identifier assigned by publisher in response to subscription request. Shall be universally unique. Shall begin with uuid:. Defined by UPnP vendor. See clause 1.1.4, "UUID format and recommended generation algorithms" for the mandatory UUID format.

#### TIMEOUT

(No TIMEOUT header field is used to cancel an event subscription.)

To cancel a subscription, a publisher shall send a response in the following format within 30 seconds, including expected transmission time.

```
HTTP/1.1 200 OK
```

### Response line

#### HTTP/1.1

The highest version supported by the origin server that is compatible with the control point that issued the request.

For example, if the control point specified support for HTTP/1.0 in the request, the response shall contain HTTP/1.0.

200 OK

HTTP defined status code indicating that no HTTP errors were detected.

If there is an error with the cancellation request, the publisher shall send a response with one of the following errors. The response shall be sent within 30 seconds, including expected transmission time.

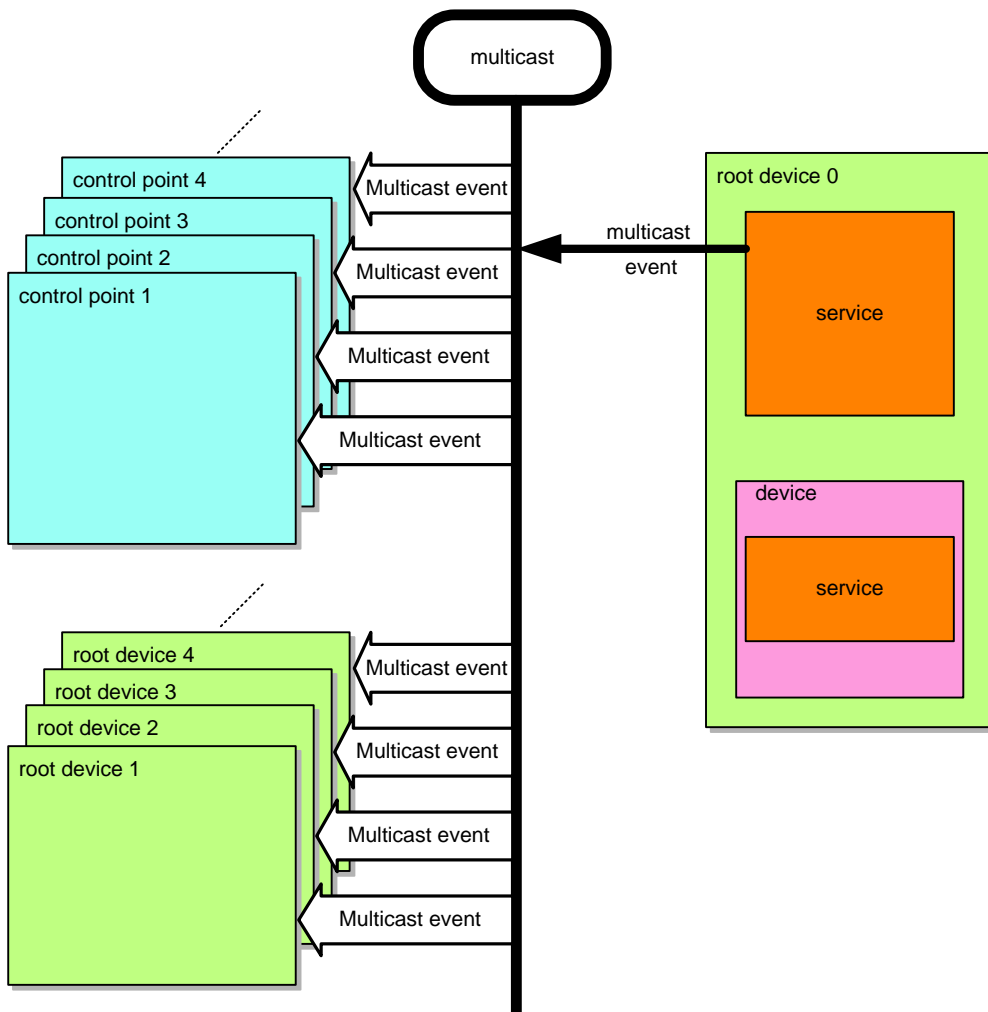
**Table 4-6: — HTTP Status Codes indicating a Cancel Subscription Error**

ErrorCode	errorDescription	Description
400	Incompatible header fields	An SID header field and one of NT or CALLBACK header fields are present.
412	Precondition Failed	An SID does not correspond to a known, un-expired subscription; or the SID header field is missing or empty.

Other errors, including other HTTP status codes, MAY be returned by layers in the protocol stack below the UPnP protocols. Consult documentation on those protocols for details.

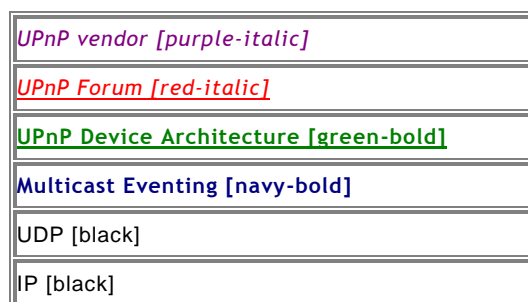
#### 4.2 Multicast Eventing

**Figure 4-3: — Multicast eventing architecture**



The publisher MAY note changes to state variables by sending *multicast event messages*. Multicast event messages contain the names of one or more state variables and the current value of those variables, expressed in XML. To send and receive multicast event messages, control points and services use the following subset of the overall UPnP protocol stack. (The overall UPnP protocol stack is listed at the beginning of this document.)

**Figure 4-4: — Multicast eventing protocol stack**



At the highest layer, multicast event messages contain vendor-specific information like vendor-specific state variable or specific variable values. Moving down the stack, vendor content is supplemented by information from a UPnP Forum working committee, such as service identifiers or variable names. Messages from the layers above are hosted in UPnP-specific protocols to transport events in a similar format to unicast UPnP events, but over a multicast address where subscriptionless eventing fits the desired usage. These messages are based on the HTTP protocol header and body format, but are not HTTP compliant because they are defined over UDP sockets. Throughout this clause, the same formatting and extension rules for SSDP as set forth in clause 1.1.2, “SSDP message header fields” and clause 1.1.3, “SSDP header field extensions” are used to give HTTP-like header field formatting. In addition, services that use evented complex datatypes shall follow the requirements in clause 2.5, “Service description”. Lastly, like SSDP, to limit network congestion, the time-to-live (TTL) of each IP packet for each multicast message should default to 2 and should be configurable. This should be the same value as that used in SSDP. When the TTL is greater than 1, it is possible for multicast messages to traverse multiple routers; therefore control points and devices using non-AutoIP addresses shall send an IGMP Join message so that routers will forward multicast messages to them (this is not necessary when using an Auto-IP address since packets with Auto-IP addresses will not be forwarded by routers).

Multicast eventing is inherently unreliable since it is based on UDP. In addition, there will be a greater possibility of message loss with greater packet size. To increase the probability of successful transmission, each message MAY be retransmitted one or more times. Therefore, UPnP working committees shall specify the event size and event retransmission rules, based on their need for reliability.

### 4.3 Event messages

A service publishes changes to certain state variables by sending event messages. These messages contain the names of one or more state variables and the current value of those variables. Event messages shall be sent in a timely manner so that subscribers are accurately informed about the state of the service and can provide a responsive user interface. If the value of more than one variable is changing at the same time, the publisher should bundle these changes into a single event message to reduce processing and network traffic.

As explained above, an initial event message is sent when a subscriber first subscribes; this event message contains the names and values for all evented variables or individual requested variables and allows the subscriber to initialize its model of the state of the service. This message should be sent as soon as possible after the publisher accepts a subscription. This message shall be sent, even if the control point unsubscribes before the message is delivered. Subscription does not cause multicast event messages.

Multicast event messages are constrained to being transported in a single UDP payload. This consideration is important when identifying variables that are to be multicast. If the cumulative size of the variables that are eligible for being sent by multicast exceeds the UDP packet's capacity, it may be necessary to send multiple, distinct multicast events.

Both unicast and multicast event messages are tagged with an event key. In unicast eventing, a separate event key shall be maintained by the publisher for each subscription to facilitate error detection (as explained below). The event key for a subscription is initialized to 0 when the publisher sends the initial event message. For each subsequent event message, the publisher increments (by one) the event key for a subscription, and includes that updated key in the event message. The event key for multicast events is also initialized to 0 when the publisher sends the initial event message. For each subsequent multicast event message, the publisher increments (by one) the event key for the multicast events, and includes that updated key in the event message. Any implementation of event keys shall handle overflow and wrap the event key from 4294967295 back to 1 (not 0). Unicast subscribers and multicast receivers shall also handle this special case when the next event key is not an increment of the previous key. The key shall be implemented as a 4 Byte (32 bit) unsigned integer.

All UPnP event messages shall be encoded using UTF-8.

#### **4.3.1 Error Cases**

For unicast eventing, the publisher shall send all event messages to the subscriber until the subscription expires even when the subscriber fails to respond. When a subscriber has missed one or more event messages, the subscriber MAY synchronize with the device's evented state by unsubscribing and re-subscribing. By doing so, the subscriber will get a new subscription identifier, a new initial event message, and a new event key.

For multicast eventing, since UDP is inherently unreliable, retransmission of a multicast event message (using the same SEQ field value) can increase the reliability. The receiver shall interpret the same SEQ field value from separate multicast event messages from a same service (identified by USN field value) as being the exactly the same message sent multiple times and shall therefore ignore such duplicates. Some state variables may change value too rapidly for some environments, for example enterprises. Working committees shall specify traffic constraints for the DCP given these concerns and guidelines. Working committees should consider both the interval for transmission of multicast events per event type (LVL:) and the retransmission rules for particular event instances.

#### **4.3.2 Unicast eventing: Event messages: NOTIFY**

To send an event message, a publisher shall send a request with method NOTIFY using the following format. The following two examples illustrate an event message using the CONTENT-LENGTH header and an event message using chunked encoding. Values in *italics* are placeholders for actual values.

Event messages sent to different subscribers that have the same sequence number shall contain the same content except for the HOST header field. A multi-homed device shall send the event message using the same pair of UPnP-enabled interfaces used for the initial subscription.

Note: XML namespace prefixes do not have to be the specific examples shown below (e.g., "s" or "u"); they can be any value that obeys the rules of the general XML namespace mechanism; control points shall accept event messages that use other legal XML namespace prefixes.

## Event message using the CONTENT-LENGTH header – Valid with HTTP/1.0 or HTTP/1.1

```
NOTIFY delivery path HTTP/1.0
HOST: delivery host:delivery port
CONTENT-TYPE: text/xml; charset="utf-8"
NT: upnp:event
NTS: upnp:propchange
SID: uuid:subscription-UUID
SEQ: event key
CONTENT-LENGTH: bytes in body

<?xml version="1.0"?>
<e:propertyset xmlns:e="urn:schemas-upnp-org:event-1-0">
  <e:property>
    <variableName>new value</variableName>
  </e:property>
  Other variable names and values (if any) go here.
</e:propertyset>
```

## Event message using chunked encoding – Valid with HTTP 1.1 only

```
NOTIFY delivery path HTTP/1.1
HOST: delivery host:delivery port
CONTENT-TYPE: text/xml; charset="utf-8"
NT: upnp:event
NTS: upnp:propchange
SID: uuid:subscription-UUID
TRANSFER-ENCODING: "chunked"
SEQ: event key

Length of chunk 1 in hexadecimal notation

<?xml version="1.0"?>
<e:propertyset xmlns:e="urn:schemas-upnp-org:event-1-0">
  <e:property>
    <variableName>new value</variableName>
  </e:property>
  Other variable names and values (if any) go here.
</e:propertyset>
0
```

Listed below are details for the request line, header fields, and body elements appearing in the listing above. Field names are not case sensitive. All field values are case sensitive except where noted. All body elements and attributes are case sensitive; body values are not case sensitive except where noted. Except where noted, required elements shall occur exactly once (no duplicates), and recommended or allowed elements MAY occur at most once. In particular, a single **propertyset** element shall not include more than one **property** element that specifies the same **variableName** element; separate event notification messages shall be used.

### Request line

#### NOTIFY

Method to notify client about event.

#### *delivery path*

Path component of delivery URL (CALLBACK header field in subscription message). Destination for event message. Single, absolute path (see also RFC 2616). shall be from one of the URLs contained in the CALLBACK header field, without truncation or modification.

#### HTTP/1.1

Highest HTTP version supported by the device. (Note: chunked encoding shall not be used if the control point supports only HTTP 1.0).

### Header fields

#### HOST

Required. Field value contains domain name or IP address and optional port components of delivery URL (CALLBACK header field in subscription message). If the port is missing or empty, port 80 is assumed.

#### ACCEPT-LANGUAGE

(No ACCEPT-LANGUAGE header field is used in event messages.)

#### CONTENT-LENGTH

Required if Origin Server does not close the session after sending the response AND Origin Server does not send the response using chunked encoding.

PROHIBITED if Origin Server sends the response using chunked encoding. Allowed otherwise.

Field value specifies the length of the body in bytes. Integer.

#### TRANSFER-ENCODING

Allowed for HTTP/1.1 and above. Field value specifies whether the response is chunked encoded by having field value "chunked" (in the example above the body is sent in a single chunk). shall not be specified if CONTENT-LENGTH header field is present.

#### CONTENT-TYPE

Required. Field value shall be "text/xml; charset="utf-8" "

#### NT

Required. Field value contains Notification Type. shall be **upnp:event**.

#### NTS

Required. Field value contains Notification Sub Type. shall be **upnp:propchange**.

#### SID

Required. Field value contains Subscription Identifier. shall be universally unique. shall begin with uuid:. Defined by UPnP vendor. See clause 1.1.4, "UUID format and recommended generation algorithms" for the MANDATORY UUID format.

#### SEQ

Required. Field value contains Event Key. Shall be 0 for initial event message. Shall be incremented by 1 for each event message sent to a particular subscriber. To prevent overflow, shall be wrapped from 4294967295 to 1. 32-bit unsigned value represented as a single decimal integer without leading zeroes (some implementations MAY include leading zeroes, which should be ignored by the recipient).

### Body

<propertyset>

Required. xmlns namespace attribute shall be urn:[schemas-upnp-org:event-1-0](#). Contains the following sub element:

  <property>

    Required. Repeat once for each variable name and value in the event message. Shall be qualified by the namespace prefix defined in the xmlns attribute of the <propertyset> element. Contains the following sub element:

      <variableName>

        Required. Element is name of a state variable that changed (<[name](#)> sub element of <[stateVariable](#)> element in service description). Shall not be qualified with any namespace. Value is the new value for this state variable. Case sensitive. Single data type as specified by UPnP service description.

For future extensibility and according to the requirements in clause 2.7, "Non-standard vendor extensions" and clause 2.8, "UPnP Device Schema", when processing XML like the listing above, devices and control points shall ignore: (a) any unknown elements and their sub elements or content, and (b) any unknown attributes and their values. Note that when subscribing to eventing with a service that is of a higher version than what is supported by the control point, event notifications MAY be sent by the service to the control point containing state variable names that are not recognized by the control point. The control point should discard and ignore such unrecognized state variables within event notification messages.

When the new value of any variable contains one or more characters reserved as markup (such as ampersand (“&”) or less than (“<”)), the text shall be escaped in accordance with the provisions of clause 2.4 of the XML specification and each such character replaced with the equivalent numeric representation or string (such as “&amp;” or “&lt;”). Such characters appearing in URLs that appear as values MAY also be percent-encoded in accordance with the URL percent-encoding rules specified in clauses 2.1 and 2.4 of RFC 3986.

On a multi-homed device, all fully-qualified URLs contained in event body that refer to resources on the device shall be reachable on the UPnP-enabled interface on which the event message is sent.

Subject to the constraints defined in clause 2.7, “Non-standard vendor extensions” and clause 2.8, “UPnP Device Schema”, control points and devices shall ignore any XML comments or XML processing instructions embedded in UPnP event messages that they do not understand. Note that because HTTP 1.1 allows use of chunked encoding, some devices MAY send the event notification using chunked encoding if the SUBSCRIBE request specified HTTP 1.1. It is therefore recommended that all implementations that include HTTP 1.1 in the SUBSCRIBE request support receiving chunked encoding.

To acknowledge receipt of this event message, a subscriber shall respond within 30 seconds, including expected transmission time. A multi-homed subscriber shall send the response using the same pair of UPnP-enabled interfaces used for the event message. If a subscriber does not respond within 30 seconds, or if the publisher is unable to connect to the subscription URL, the publisher should abandon sending this message to the subscriber but shall keep the subscription active and send future event messages to the subscriber until the subscription expires or is cancelled. The subscriber shall send a response in the following format.

```
HTTP/1.1 200 OK
```

### Response line

HTTP/1.1

Highest HTTP version supported by the control point that is compatible with the device that sent the event message.

200 OK

HTTP defined status code indicating that no HTTP errors were detected.

(No body for a request with method NOTIFY, but note that the message shall have a blank line following the last HTTP header field.)

If a device sends an event to a control point using HTTP/1.0 without the KeepAlive token, the control point shall close the socket after responding. If a device sends an event to a control point using HTTP/1.1 and sets the Connection:CLOSE token, the control point shall close the socket after responding.

If there is an error with the event message, the subscriber shall respond with one of the following errors. The response shall be sent within 30 seconds, including expected transmission time.

**Table 4-7: — HTTP Status Codes indicating a Notify Error**

ErrorCode	errorDescription	Description
400	Bad request	The NT or NTS header field is missing; or the request is malformed.

ErrorCode	errorDescription	Description
412	Precondition Failed	An SID does not correspond to a known, un-expired subscription; or the NT header field does not equal <a href="#">upnp:event</a> ; or the NTS header field does not equal <a href="#">upnp:propchange</a> ; or the SID header field is missing or empty.

Other errors, including other HTTP status codes, MAY be returned by layers in the protocol stack below the UPnP protocols. Consult documentation on those protocols for details.

### 4.3.3 Multicast Eventing: Event messages: NOTIFY

To send a multicast event message, a publisher shall send a message with method NOTIFY using the following format. The following example illustrates an event message using the CONTENT-LENGTH header. Values in *italics* are placeholders for actual values.

A multi-homed publisher shall multicast the event message on each of its UPnP-enabled interfaces. Event messages sent on different UPnP-enabled interfaces that have the same sequence number shall contain the same content except for possibly the HOST header field and any fully-qualified URLs contained in the event body. The HOST header field of an advertisement shall be the standard multicast eventing address specified for the protocol (IPv4 or IPv6) used on the interface. All fully-qualified URLs contained in the event body that refer to resources on the device shall be reachable on the UPnP-enabled interface on which the event message is sent.

Note: XML namespace prefixes do not have to be the specific example shown below (e.g., “s” or “u”); they can be any value that obeys the rules of the general XML namespace mechanism; control points shall accept event messages that use other legal XML namespace prefixes.

#### Multicast event message using the CONTENT-LENGTH header – Valid with HTTP/1.0 or HTTP/1.1

```

NOTIFY * HTTP/1.0
HOST: 239.255.255.246:7900 *** note the port number is different than SSDP ***
CONTENT-TYPE: text/xml; charset="utf-8"
USN: Unique Service Name for the publisher
SVCID: ServiceID from SCPD
NT: upnp:event
NTS: upnp:propchange
SEQ: monotonically increasing sequence count
LVL: event importance
BOOTID.UPNP.ORG: number increased each time device sends an initial announce or update message
CONTENT-LENGTH: bytes in body <?xml version="1.0"?>
<e:propertyset xmlns:e="urn:schemas-upnp-org:event-1-0">
  <e:property>
    <variableName>new value</variableName>
  </e:property>
  <!-- Other variable names and values (if any) go here. -->
</e:propertyset>

```

Listed below are details for the request line, header fields, and body elements appearing in the listing above. Field names are not case sensitive. All field values are case sensitive except where noted. All body elements and attributes are case sensitive; body values are not case sensitive except where noted. Except where noted, required elements shall occur exactly once (no duplicates), and recommended or allowed elements MAY occur at most once. In particular, a single **propertyset** element shall not include more than one **property** element that specifies the same **variableName** element; separate event notification messages shall be used.

#### Request line

Shall be “NOTIFY \*HTTP/1.1”

#### Header fields



HOST

Required. Field value shall be [239.255.255.246:7900](#). Please note that port number [7900](#) is different from SSDP port number [1900](#).

CONTENT-LENGTH

Required. Field value specifies the length of the body in bytes. Integer. Chunked encoding shall not be used for multicast event messages.

CONTENT-TYPE

Required. Field value shall be ["text/xml; charset=utf-8"](#).

USN

Required. Field value contains Unique Service Name for the publisher. Identifies a unique instance of a service in a unique instance of a device. It shall be one of the following forms. The prefix (before the double colon) shall match the value of the UDN element in the device description. (Clause 2, "Description" explains the UDN element.) Single URI.

[uuid:device-UUID::urn:schemas-upnp-org:service:serviceType:ver](#)

where device-UUID is specified by the UPnP vendor; serviceType and ver are defined by the UPnP Forum working committee. See clause 1.1.4, "UUID format and recommended generation algorithms" for the MANDATORY UUID format.

[uuid:device-UUID::urn:domain-name:service:serviceType:ver](#)

where device-UUID, domain-name, serviceType and ver are defined by the UPnP vendor. See clause 1.1.4, "UUID format and recommended generation algorithms" for the MANDATORY UUID format. Period characters in the domain name shall be replaced by hyphens in accordance with RFC 2141.

SVCID

Required. Field value contains ServiceID from the SCPD to uniquely identify which service generated the event. As defined in clause 2.2, "Generic requirements on XML usage"

NT

Required. Field value contains Notification Type. Shall be [upnp:event](#).

NTS

Required. Field value contains Notification Sub Type. Shall be [upnp:propchange](#).

SEQ

Required. Field value contains Event Key. The numeric sequence count shall be 0 for initial multicast event message. Shall be incremented by 1 for each multicast event message per a service; however, when a multicast message is retransmitted, it shall be sent with its original event key. To prevent overflow, shall be wrapped from 4294967295 to 1. 32-bit unsigned value represented as a single decimal integer without leading zeroes (leading zeroes, if present, shall be ignored by the recipient).

LVL

Required. Field value shall be a string in UTF-8. Event level allows the receiver to first level filter messages based on the value and is defined by the UPnP Technical Committee. See Table 4-8, "Multicast event levels" for the Event Levels defined with this version of the UPnP architecture. UPnP Working Committees shall specify event level values when defining events that will be multicast.

The following table summarizes defined event levels and the expected meaning of those values. Event levels defined by the UPnP Forum Technical Committee start with the prefix "upnp:". Vendor and other extensions outside the UPnP Forum shall be prefixed by the domain name of the defining organization. For example: "domain.org/alerts/level/"

**Table 4-8: — Multicast event levels**

Event Level	Description
<a href="#">upnp:/emergency</a>	The event carries critical information that the device should act upon immediately.
<a href="#">upnp:/fault</a>	The event carries information related to an error case

Event Level	Description
<a href="#">upnp:/warning</a>	The event carries information that is a non-critical condition that the device MAY want to process or pass to the user
<a href="#">upnp:/info</a>	The event carries information about the normal operation of the device that may be of interest to end-users. This information is simply informative and does not indicate any abnormal condition or status such as a warning or fault. Other event levels are defined for those purposes.
<a href="#">upnp:/debug</a>	The event carries debug information typically used by programmers and test engineers to evaluate the internal operation of the device. This information is typically not displayed to end users.
<a href="#">upnp:/general</a>	For events that fit into no other defined category
<code>&lt;domain&gt;:/&lt;level&gt;</code>	Example vendor extension. Domain is the ICANN domain name for the vendor and level is an arbitrary string defined by the vendor. E.g. domain.org:/alerts/type/

BOOTID.UPNP.ORG

Required. As defined in clause 1.2, and 1.2.2.

## Body

`<propertyset>`

Required. xmlns namespace shall be “urn:[schemas-upnp-org:event-1-0](#)”. Contains the following sub element:

`<property>`

Required. Repeat once for each variable name and value in the event message. Shall be qualified by the namespace prefix defined in the `xmlns` attribute of the `<propertyset>` element. Contains the following sub element:

`<variableName>`

Required. Element is name of a state variable that changed ([name](#) sub element of [stateVariable](#) element in service description). Shall not be qualified with any namespace. Value is the new value for this state variable. Case sensitive. Single data type as specified by UPnP service description.

Note that for simplicity many of the header fields for multicast eventing are the same as for unicast eventing. These include: HOST, CONTENT-TYPE, USN, NT, NTS, and SEQ. In addition, the body of the message (propertyset) has the same format as unicast events.

For future extensibility and according to the requirements in clause 2.7, “Non-standard vendor extensions” and clause 2.8, “UPnP Device Schema”, when processing XML like the listing above, devices and control points shall ignore: (a) any unknown elements and their sub elements or content, and (b) any unknown attributes and their values. Subject to the constraints defined in clause 2.7, “Non-standard vendor extensions” and clause 2.8, “UPnP Device Schema”, control points and devices shall ignore any XML comments or XML processing instructions embedded in UPnP device and service descriptions that they do not understand. The control point should discard and ignore unrecognized state variables within multicast event notification messages.

When the new value of any variable contains one or more characters reserved as markup (such as ampersand (“&”) or less than (“<”)), the text shall be escaped in accordance with the provisions of clause 2.4 of the XML specification and each such character replaced with the equivalent numeric representation or string (such as “&amp;” or “&lt;”). Such characters appearing in URLs that appear as values MAY also be percent-encoded in accordance with the URL percent-encoding rules specified in clauses 2.1 and 2.4 of RFC 3986.

#### 4.4 UPnP Event Schema

The UPnP Event Schema defines the structures and data types used in the body of UPnP event notifications. As explained with the UPnP Device and Service Schemas, the UPnP Event Schema is written in XML syntax according to the conventions of XML Schema (Part 1: Structures, Part 2: Datatypes). The UPnP Event Schema is defined within a UPnP service template; however, the schema shall conform to the format as defined in clause B.5, “UPnP Event Schema”. The elements it defines are used in event notifications.

As explained in clause 2, “Description”, the UPnP Service Schema also specifies a `sendEvents` attribute for a state variable. The default value for this attribute is “yes”. To denote that a state variable is evented, the value of this attribute is “yes” (or the attribute is omitted) in a service description; to denote that a state variable is non-evented, the value is “no”. Note that if all of a service’s state variables are non-evented, the service has nothing to publish, and control points cannot subscribe and will not receive event messages from the service.

#### 4.5 Augmenting the UPnP Device and Service Schemas

Some state variables may change value too rapidly for eventing to be useful. UPnP Forum Working Committees or UPnP vendors may document moderation in the number of event messages sent due to changes in a variable’s value. Event moderation may include limitation on the frequency in reporting change of a state variable value or a minimum degree of change that shall occur before a change is reported.

Parameter	Description
<b>maximumRate</b>	Single numeric value (in seconds) of type integer or float. State variable <i>v</i> shall not be part of an event message more often than every <i>n</i> seconds. If <i>v</i> is the only state variable changing, then an event message containing the state variable shall not be generated more often than every <i>n</i> seconds. If <i>v</i> has changed sooner than <i>n</i> seconds from the last event message that contains <i>v</i> , then an event message containing the current value of <i>v</i> shall be sent in a timely manner after <i>n</i> seconds from the previous event message containing <i>v</i> . If <i>v</i> has not changed within <i>n</i> seconds following the last event message that contains <i>v</i> , then when <i>v</i> does change an event message with the current value of <i>v</i> shall be sent in a timely manner. Specifying a <b>maximumRate</b> value is useful for variables that model frequently changing state variables.
<b>minimumDelta</b>	Single numeric value (minimum change required) whose type shall match the corresponding state variable. State variable <i>v</i> shall not be part of an event message unless its value has changed (plus or minus) by at least <b>minimumDelta</b> since the last time an event message was sent that contains <i>v</i> . Only valid for state variables with a numeric (integer or float) data type. Specifying a <b>minimumDelta</b> value is useful for variables that model continuously changing state variables.

The publisher MAY send out any changed moderated variable when an event goes out. The publisher shall meet moderation rules described above, but the publisher MAY flush recent changes when it sends out an event message.

Note that moderation affects events only and not state table updates. Specifically, control actions which return the value of state variables MAY return a more current value than published via eventing. Put another way, moderation means that not all state table changes result in events.

Decisions about which variables to event and any possible moderation is up to the appropriate UPnP Forum working committee (for standard services) or a UPnP vendor (for non-standard services).

#### 4.6 References

RFC 2616, HTTP: Hypertext Transfer Protocol 1.1. Available at:  
<http://www.ietf.org/rfc/rfc2616.txt>.

RFC 3986, Uniform Resource Identifiers (URI): Generic Syntax. Available at:  
<http://www.ietf.org/rfc/rfc3986.txt>.

XML, Extensible Markup Language. Available at: <http://www.w3.org/TR/2000/REC-xml-20001006>.

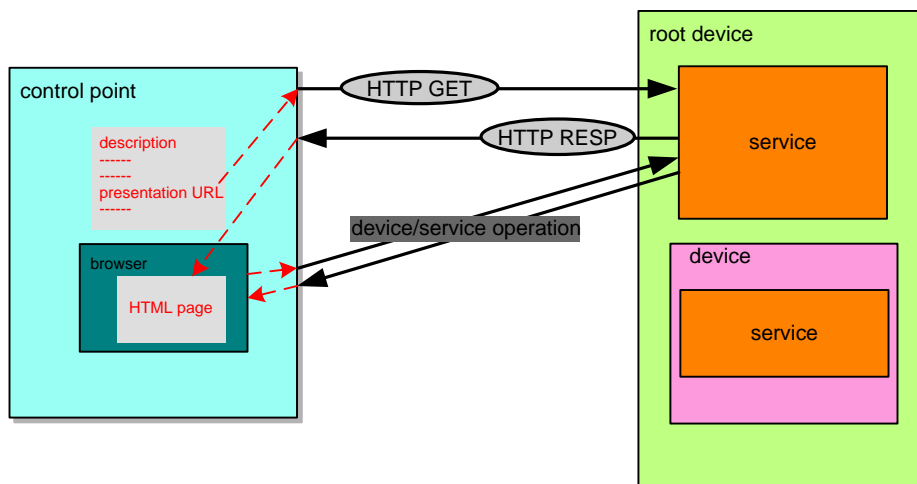
XML Schema (Part 1: Structures, Part 2: Datatypes). Available at: <http://www.w3.org/TR/xmlschema-1>, <http://www.w3.org/TR/xmlschema-2>.

## 5 Presentation

*Presentation is Step 5 in UPnP networking. Presentation comes after addressing (Step 0) where devices get network addresses, after discovery (Step 1) where control points find interesting device(s), and after description (Step 2) where control points learn about device capabilities. Presentation exposes an HTML-based user interface for controlling and/or viewing device status. Presentation is complementary to control (Step 3) where control points send actions to devices, and eventing (Step 4) where control points listen to state changes in device(s).*

After a control point has (1) discovered a device and (2) retrieved a description of the device, the control point is ready to begin presentation. If a device has a URL for presentation, then the control point can retrieve a page from this URL, load the page into a browser and, depending on the capabilities of the page, allow a user to control the device and/or view device status. The degree to which each of these can be accomplished depends on the specific capabilities of the presentation page and device.

**Figure 5-1: — Presentation architecture**



The URL for presentation is obtained from the presentationURL element in the device description. If presentationURL is an absolute URL, the fully qualified presentation URL is the presentationURL. Otherwise, if presentationURL is a relative URL, the fully qualified presentation URL is the URL resolved from presentationURL in accordance with clause 5 of RFC 3986, using either the URLBase element, if specified, or the URL from which the device description was retrieved as the base URL. A multi-homed control point that attempts to retrieve a presentation page on a particular UPnP-enabled interface shall use the fully qualified presentation URL from the description document received on that interface. The device description is delivered via a description message. Clause 2, “Description” explains the device description and description messages in detail.

Retrieving a presentation page is a simple HTTP-based process and uses the following subset of the overall UPnP protocol stack. (The overall UPnP protocol stack is listed at the beginning of this document.)

**Figure 5-2: — Presentation protocol stack**

<i>UPnP vendor</i> [purple-italic]
<b>UPnP Device Architecture</b> [green-bold]
HTTP [black]
TCP [black]
IP [black]

At the highest layer, the presentation page is specified by a UPnP vendor. Moving down the stack, the UPnP Device Architecture specifies that this page be written in HTML. The page is delivered via HTTP over TCP over IP. For reference, colors in [square brackets] are included for consistency with other clauses in this document.

To retrieve a presentation page, the control point issues an HTTP GET request to the presentation URL, and the device returns a presentation page. Responses to HTTP GET requests for presentation pages shall be sent using the same address on the same interface on which the HTTP GET was received. The generic requirements on HTTP usage in UPnP 2.0 (as defined in clause 2.1, “Generic requirements on HTTP usage” of this document) shall be followed by devices and control points that implement presentation.

Unlike the UPnP Device and Service Templates, and standard device and service types, the capabilities of the presentation page are completely specified by the UPnP vendor. The page shall be an HTML page; it is recommended that the page be based upon XHTML-Basic. However, other design aspects are left to the vendor to specify. This includes, but is not limited to, all capabilities of the control point's browser, scripting language or browser plug-ins used, and means of interacting with the device. To implement a presentation page, a UPnP vendor MAY wish to use UPnP mechanisms for control and/or eventing, leveraging the device's existing capabilities but is not constrained to do so.

Presentation pages should use mechanisms provided by HTML for localization (e.g., META tag with charset attribute). Control points should use the ACCEPT-LANGUAGE and CONTENT-LANGUAGE feature of HTTP to try to retrieve a localized presentation page. Specifically, a control point MAY include a HTTP ACCEPT-LANGUAGE header field in the request for a presentation page; if an ACCEPT-LANGUAGE header field is present in the request, the response shall include a CONTENT-LANGUAGE header field to identify the page's language.

It is recommended that fully qualified URLs to resources on the device are not embedded in HTML presentation pages, but that relative URLs are used instead, so that the host portion of the embedded URLs does not need to be modified when sent on different UPnP-enabled interfaces.

## 5.1 References

RFC 3986, Uniform Resource Identifiers (URI): Generic Syntax. Available at: <http://www.ietf.org/rfc/rfc3986.txt>.

HTML, HyperText Markup Language. Available at: <http://www.w3.org/TR/html4>.

XHTML™ Basic. Available at: <http://www.w3.org/TR/xhtml-basic/>.

## **Annex A** **(normative)** **IP Version 6 Support**

### **A.0 Note (informative)**

2014 IPv6 Support Update: This revision to the UPnP Device Architecture Annex A is motivated by a need to allow UPnP services and implementations to determine address selection preferences, and not require they abide by the default RFC 6724 preference scheme. RFC 6724 (which obsoleted RFC 3484) expressed IPv6 preference as a default behavior, but explicitly recognized that applications may have good reason to use other preference schemes, and described several possible other preference schemes.

2012 IPv6 Support Update: This revision to the UPnP Device Architecture Annex A is motivated by operational experience gained since the previous update. There are four high level goals addressed with these latest changes:

1. IPv6 support needs to be required in order for new and existing devices to interoperate with each other
2. GUA and ULA should be treated with equal weight, with address selection performed according to IETF guidance
3. IPv6 needs to be preferred over IPv4 whenever it is available, per IETF guidance
4. The IPv6 updates need to be applicable to UDA 1.0, UDA 1.1 and UDA 2.0 implementations

UPnP devices and control points need to be encouraged to support IPv6. The IPv4 public address space is effectively exhausted, and the move towards IPv6 is considered inevitable. Service providers and on-line content providers are all adding support of IPv6. Consumer electronics devices that are capable of accessing on-line services and content will also need to move to IPv6. In fact, the IETF has signaled its opinion on the deployment of IPv6 in RFC 6540, titled "IPv6 Support Required for all IP-capable Nodes." Also, as IPv6 standards and support progress, there may be additional benefit to having UPnP supported over IPv6. There are potential advantages to the Remote Access architecture, as well as possible future benefits for traversal of multiple routers inside a home network. IPv6 support is being encouraged by CEA through the efforts of a new CEA Working Group focused on IPv6 and the transition from IPv4. UPnP needs to stay relevant in the evolving IP world, which demands that IPv6 interoperability be enabled. Legacy UDA 1.0 certified devices are allowed to remain non-IPv6 compliant, however it is highly recommended that all UPnP devices and control points support IPv6 and dual-stack operation.

While it may be preferable to use ULA for communication within the home network, preferring ULA over GUA under all conditions would result in breaking external connectivity. This revision of Annex A defines ULA and GUA as having equal weight. It describes a "use both" approach to discovery and advertisement where devices and control points perform initial communication on all enabled IP addresses (IPv4 and IPv6 alike). For other communication, there are standard mechanisms for making address selection that do not require a default overriding preference for one or the other addressing schemes. These methods are documented in IETF RFC 3484, which is currently being updated. UPnP devices that follow the most up-to-date standard for address selection will have the highest guarantee of interoperability. Note that IETF RFC 3484 also requires devices to prefer IPv6 over IPv4. There are a variety of reasons why the IETF has taken this approach, and these reasons apply to UPnP devices and applications just as well as other devices and applications. It is also desirable for UPnP devices and applications to use address selection mechanisms that are consistent with those used by other applications and devices.

### **A.1 Introduction**

Most of today's Internet uses IPv4, which was first standardized in 1981. IPv4 has been remarkably resilient in spite of its age, but it is beginning to have problems. Most importantly, there is a growing shortage of IPv4 addresses, which are needed by all new machines added to the Internet. Deployment of large numbers of UPnP devices will only exacerbate the shortage.

IPv6 overcomes many shortcomings of the IPv4 protocol, such as the comparatively limited number of available addresses and provides other significant optimizations. It also adds many improvements to IPv4 in areas such as routing and network automatic configuration. IPv6 is expected to gradually replace IPv4, with the two coexisting for a number of years, if not decades, during a transition period.

This Annex describes mechanisms that will allow devices and control points based on the UPnP Device Architecture to be used on IPv6 networks.

## A.2 General Principles

### A.2.1 UPnP Device Architecture V1.0

UDA 1.0 devices and control points SHALL implement IPv4 and support IPv4-only operation, and SHOULD implement IPv6 and support dual stack (IPv4 and IPv6) operation. Vendors MAY choose to allow IPv6-only operation as a policy alternative, but a device and a control point SHALL run IPv4 in order to be certified.

### A.2.2 UPnP Device Architecture V2.0

UDA 2.0 devices and control points SHALL implement IPv4 and IPv6, SHALL support dual stack (IPv4 and IPv6) operation, and SHALL be capable of operating on IPv4-only and IPv6-only networks. Vendors MAY choose to allow IPv4-only and/or IPv6-only operation as configuration alternatives, but a device and a control point SHALL run dual-stack in order to be certified.

### A.2.3 IPv6 and Dual Stack

Devices operating in dual-stack mode SHALL perform discovery and advertisement on all available non-temporary (RFC 4941) Globally Unique Addresses (GUA), Unique Local Addresses (ULA), Link-Local Addresses, and IPv4 addresses. This list excludes temporary addresses, deprecated address space, and addresses associated with various IPv4 to IPv6 transition technologies (e.g., 6to4, Teredo, ISATAP). While not expressly forbidden, these addresses SHOULD NOT be used for discovery and advertisement. IETF RFC 6724 recommends default source and destination address-selection behavior but specifically notes that its selection rules are not to be construed to override an application or upper layer's explicit choice of a legal destination or source address. UPnP control points and devices SHOULD use RFC 6724 default destination address selection algorithms in the absence of more specific guidance. However, it is allowed for more specific guidance to exist, and UPnP implementers are encouraged to identify if such guidance is needed to meet their particular use cases. For any selected destination address, control points and devices SHALL use RFC 6724 source address selection algorithms.

All IPv6 devices and control points that implement both stacks are inherently multi-homed. An IPv6 device or control point MAY also have multiple IPv6 addresses from both ULA and GUA address scopes. The basic principle is that the unicast addresses advertised by a device are consistent with the scope used for that multicast advertisement message. The rules for associating an IPv6 address with a particular multicast scope are defined based on the degree of "routeability" provided by a particular IPv6 address. The two multicast scopes currently defined for use in UPnP are:

- Link-Local Scope (the stations reachable without routing) which uses addresses called Link-Local Addresses.
- Site-Local Scope (a private network consisting of one or more links bounded by a site's administrative edge) which uses Unique Local Addresses (ULA) or Globally Unique Addresses (GUA). This scope is inclusive of the Link-Local Scope.

See RFC 4291 Section 2.7 for multicast scope definitions.

The Internet Assigned Numbers Authority has registered a multicast address and port for SSDP: an address is of the form FF0X::C. This is a variable scope multicast address where X is changed to represent the appropriate scope. A device advertising on the local link would use a scope of 2 and address [FF02::C]:1900. A site-scope advertisement on the home network would use scope 5 [see 2.7 of RFC 4291] and specify address [FF05::C]:1900. Table A-1 shows the multicast scope to use with a particular source address.

Since a UPnP device uses multicast for advertisements, and multicast eventing, and there is a corresponding scope defined for multicast, the rules set forth below define the selection of unicast addresses in the context of the multicast scope used for multicast messages. In addition, a UPnP device SHALL adhere to all multi-homed behaviors described in this document.

The following requirements apply to devices and control points using the UPnP Device Architecture over IPv6. This is an overview of the process. Implementers SHALL refer to the cited references for details and conform to requirements included within those citations.

- a) Devices and control points SHALL use only the Link-Local unicast address as the source address and when specifying a literal IP address in LOCATION URLs in all multicast messages that are multicast to the Link-Local scope FF02::C for SSDP and FF02::130 for multicast eventing. Devices and control points SHALL listen on the Link-Local scope. See RFC 4862 for details of Link-Local addressing.
- e) Devices and control points SHALL implement SLAAC (RFC 4862) for address assignment on each IPv6-enabled interface. They MAY implement DHCPv6 with the IA-NA option (RFC 3315) for address assignment. Once the address assignment process is complete, the device or control point will have exactly one link local address available (unless Duplicate Address Detection failed) and zero, one, or more ULA and/or GUA addresses available for use, on an IPv6-enabled interface.
- f) Devices and control points SHALL use an acquired ULA or GUA in all multicast messages as the source address and when specifying a literal IP address in LOCATION URLs that are multicast to the Site-Local scope addresses of either FF05::C or FF05::130.
- g) Devices and control points SHALL NOT send Global scoped, Organization-Local scoped, or Admin-Local scoped multicast messages.
- h) A device SHALL listen for unicast SSDP traffic on all multicast scopes on which it has advertised, for each UPnP-enabled interface. Control points SHALL listen for unicast SSDP traffic in order to identify and control devices.
- i) The hop limit of each IP packet for a Site-Local scope multicast message SHALL be configurable and SHOULD default to 10.
- j) When a LOCATION URL includes a literal IP address, that IP address SHALL match the unicast source address used to send the message. When a LOCATION URL includes a fully qualified domain name (FQDN), the resolution of that FQDN SHALL include the IP address used as the unicast source address. For example, this means that if mDNS or other mechanisms are used to create DNS entries, that DNS translations will need to exist for all of a device's IP addresses that are used for discovery and advertisement.
- k) Since devices will multicast NOTIFY and M-SEARCH messages on multiple IP addresses, receiving devices SHALL consider all messages with the same USN and BOOTID.UPNP.ORG or NLS value as being from a single device and select exactly one (1) address to use in subsequent unicast communication. When multiple multicast messages are received from a single host (each with different source IP address), the receiving device SHALL select a destination address (from among the source addresses of the multiple multicast messages) and LOCATION URL to use for subsequent unicast communication with a particular device by using an algorithm that results in consistent address selection behavior. The default destination address selection rules are described in RFC 6724, but alternate algorithms can be implemented.
- l) Devices SHALL select a source address to use when responding to multicast messages according to the selection rules described in RFC 6724. Note that this will mean selecting a source address that best matches the destination address (the source address of the multicast message that the receiving device selected in the above requirement).
- m) Devices which multicast messages on multiple interfaces (i.e. IPv4 and IPv6) SHALL ensure that the transmission start time of each multicast message on corresponding interfaces is less than 20ms. Note: Compliance with this requirement will be evaluated under average network conditions. It is understood that certain non-nominal network conditions MAY prevent timely transmission on particular interfaces.



**Table A-1: — Matching of Device Address to Multicast Scope**

Device Address	Link-Local multicast	Site-Local multicast	Global or other multicast
Link (link local)	Y		Never
Site (ULA)		Y	Never
Global (GUA)		Y	Never

#### **A.2.4 Device operation**

A device supporting both IPv4 and IPv6 simultaneously SHALL be advertised using the same Unique Service Name (USN) on all IPv4 and IPv6 interfaces and SHALL have identical device description documents and service description documents when accessed from both protocols. The URLBase element of the device description document SHALL NOT be used. Dual stacked devices should consider power constraints (such as radio activity in battery powered devices) when sending multiple NOTIFY and M-SEARCH messages (e.g. grouping message transmission by service or address family). These devices SHALL also conform to other multi-homed descriptions in the respective sections of the document.

#### **A.2.5 Control point operation**

Control points SHALL use the matching USNs of multiple IPv4 and IPv6 announcements to treat multi-address devices as a single device. Dual stacked control points should consider power constraints (such as radio activity in battery powered devices) when sending multiple NOTIFY and M-SEARCH messages (e.g. queuing messages and grouping message transmission by service or address family). In addition, control points SHALL also conform to other multi-homed descriptions in the respective sections of this document.

### **A.3 Addressing**

RFC 4862, RFC 4193 and RFC 3315 describe how a physical device obtains an IPv6 address.

IPv6 multicast addresses include a component (scope) which determines the propagation of a message. The multicast scope for UPnP SHALL be Link-Local or Site-Local scoped. The Site-Local scope is encompassing of Link-Local. That is, Link-Local scope is contained in Site-Local scope.

In IPv6 networks, a Link-Local IP address is assigned per interface by the physical device, and therefore UPnP devices or control points on IPv6-enabled devices will always have a Link-Local address. In addition, a device or control point MAY or MAY NOT have a Unique Local Address (ULA) or a Global Unicast Address (GUA) available to it. IPv6 devices or control points are “multi-homed” when they run UPnP on one or more IPv6 addresses for an interface: a Link-Local address for local link traffic is always available, and a globally routable address and/or a site-routable address, using a ULA, is available when the router advertises a prefix for use in either stateless address autoconfiguration (SLAAC) or DHCPv6 supplied by the home router. In some scenarios, devices or control points MAY only have a Link-Local address available to them; reasons for this include underlying device capability, administrative policy, and availability of ULA and global prefixes. Link-Local addresses are generated by the physical device itself, without referring to an outside router or server such as a DHCPv6 server.

Thus, there are two considerations for UPnP IPv6 addressing. The first is availability: If the gateway/router does not advertise a GUA or ULA prefix, then UPnP IPv6 addressing is strictly Link-Local addressing. The second is policy: This Annex recommends that multicast messages be sent using all non-temporary GUA, ULA, Link-Local, and IPv4 addresses as a source address, that destination address selection (from among the source addresses of the multicast messages) use an algorithm that results in consistent selection behavior (default algorithm is defined in RFC 6724), and subsequent source address selection follow the source address selection procedure described in RFC 6724.

In addition to the address(es) assigned by this address selection process, each device or control point, acting as a normal IPv6 node, listens for traffic on several multicast addresses: link-local scope all-nodes multicast address FF02::1; the site scope all-nodes multicast address FF05::1 and multicast addresses of joined groups on each interface.

### **A.3.1 UPnP Messaging on IPv6 Interfaces**

At a minimum, a UPnP Device and a Control Point SHALL both listen and send on IPv6 Link-Local scope multicast and unicast addresses. A UPnP device SHALL send announcements and multicast eventing messages to, and listen for search requests on the assigned-to-UPnP Link-Local scope multicast addresses and receive connections on a Link-Local unicast address that it has advertised. A UPnP Control Point SHALL listen for announcements and multicast eventing messages on the assigned-to-UPnP Link-Local scope multicast addresses and be capable of requesting service definitions using a Link-Local unicast address.

Additionally, a UPnP Device and a Control Point SHALL be capable of both listening and sending on Site-Local scope multicast and site-routable (ULA or GUA) addresses. Whether or not a particular Device or Control Point uses Site-Local scope is a policy decision, based on the availability of site-routable addresses and on address selection policy. In order to accommodate routing across prefixes within a home network, this specification REQUIRES Site-Local scope capability in UPnP Devices and Control Points.

### **A.3.2 Summary of boot/startup process**

- For IPv4, Auto-IP addressing is performed as specified in section 0 “Addressing” of this document.
- For IPv6, address assignment is performed as specified in RFC 4862, RFC 4193, and RFC 3315. This address assignment is handled by the underlying IPv6 stack. That stack is expected to make all IPv6 addresses available to UPnP applications.

### **A.3.3 Address Selection and RFC 6724**

As described in the normative sections above, UPnP dual-stack Devices operate both at IPv6 multicast Link-Local scope and Site-Local scope, as well as IPv4 multicast, by default, and advertise their services in SSDP messages sent in Site-Local scope IPv6 multicast packets *in addition to* Link-Local scope multicast. As explained above, the default is for UPnP dual-stack devices to perform destination address selection as explained in RFC 6724 (though other algorithms can be used). Source address selection for unicast messages is based on a selected destination address and done according to RFC 6724.

## **A.4 Discovery**

The UPnP discovery phase does not substantially change when used over IPv6. All definitions of section 1 “Discovery” of this document SHALL be followed, except when a change is mentioned in this section.

IGMP is the protocol used by IPv4 to ensure that incoming multicast traffic is forwarded by a router to the network segment to which the router is attached. IGMP requires that the devices and control points attached to the network segment contact the router to notify it of their interest in certain multicast addresses. The protocol that provides this service in IPv6 is Multicast Listener Discovery protocol (MLD). UPnP Control points and devices SHALL participate in the MLD protocol (either directly, or indirectly via APIs to an IPv6 stack) for any Link-Local and Site-Local scope UPnP IPv6 multicast message. MLDv1 as specified in RFC 2710 is sufficient for UPnP, although more recent and backward-compatible versions of MLD may be implemented.

IP addresses embedded in UPnP messages and descriptions sent in response to requests received on IPv6 addresses will generally be literal addresses formatted according to RFC 3986 and RFC 5952 (including those in discovery messages, the URLBase element of the device description (if specified), and HTTP HOST header fields). In UDA 2.0, together with the USN, the BOOTID.UPNP.ORG header field allows control points to recognize when a message received on a different protocol or address is effectively the same message (in this case, the BOOTID.UPNP.ORG field value will be the same in all announcements sent with

different addresses at roughly the same time), as opposed to being a new advertisement from a device which has changed from one protocol or address to another (in which case, the BOOTID.UPNP.ORG field value will differ between the old and the new announcement).

#### A.4.1 OPT and NLS

For backward compatibility with devices and control points implementing UPnP over IPv6 according to the original provisions of Annex A to UPnP Device Architecture version 1.0; IPv6 enabled devices SHALL include an OPT header field and NLS header field and IPv6 enabled control points SHALL recognize OPT and NLS header fields. The OPT header field is defined by the HTTP Extension Framework (RFC 2774); the OPT header field is used (rather than MAN) because it is possible for a control point to function without recognizing the NLS header field, although the user experience will be suboptimal (and IPv4-only control points may not recognize NLS). The NLS field value, contains a string value which SHALL change whenever the network configuration of the device changes (e.g., if any of the assigned or calculated IP addresses change). It is recommended that a GUID (in the standard UUID text format, for example, "0000002F-0000-0000-C000-000000000046") be used for this purpose, since all UPnP devices SHALL already have the ability to generate GUIDs; however, other techniques are possible. Since under UPnP Device Architecture version 1.1 and UDA 2.0 the BOOTID.UPNP.ORG field value is required to be unique on each device reboot or configuration change, the field value of the NLS header field can be set the same as the field value of the BOOTID.UPNP.ORG header field to simplify implementation. The NLS value SHALL be at least 1 and no more than 64 characters in length.

#### A.4.2 Advertisement

For IPv6, a device advertises over IPv6 according to the following guidelines:

- SSDP announcements SHALL be sent to [FF0X::C]:1900 (with "X" being either 2 or 5 and set appropriately depending on the multicast scope upon which the announcement is being sent). Control points SHALL listen to these addresses and ports to detect when new devices are available on the network.
- SSDP announcements SHALL be sent using all non-temporary GUA, ULA, Link-Local, and IPv4 addresses. For example, a UPnP device with an IPv4 address, a Link-Local IPv6 address, a ULA and a GUA would send 4 announcements.
- As described in section 1.2.2 "Device available – NOTIFY with ssdp:alive" of this document, announcements sent over IPv6 and IPv4 SHALL use the same CACHE-CONTROL field value and SHALL be sent within a 20 ms window. When all advertisements, both over IPv4 and IPv6, have expired, the control point SHALL assume that the device (or service) is no longer available.
- The SSDP HOST field value SHALL contain an IPv6 address instead of an IPv4 address.
- The SSDP LOCATION field value contains the URL of the root device description document. Typically, a literal IPv6 address formatted according to RFC 3986 will be used. An IPv6 address SHALL be contained within brackets if a port is specified. The host address in the URL SHALL be valid within the current scope (the address or scope on which the announcement is being sent). Specifically, a device advertising over IPv6 SHALL NOT use an IPv4 address in the SSDP LOCATION header field.
- The OPT and NLS header fields SHALL be included [A.4.1].

The UDA 2.0 example below incorporates this syntax.

```
NOTIFY * HTTP/1.1
HOST: [FF02::C]:1900
CACHE-CONTROL: max-age = seconds until advertisement expires
LOCATION: URL for UPnP description of this device
OPT: "http://schemas.upnp.org/upnp/1/0/"; ns=01
01-NLS: same value as BOOTID field value
NT: notification type
NTS: ssdp:alive
SERVER: OS/version UPnP/2.0 product/version
BOOTID.UPNP.ORG: number increased each time device sends an initial announce or update
message
CONFIGID.UPNP.ORG: number used for caching description information
USN: composite identifier for the advertisement
```

### A.4.3 Advertisement: Device unavailable

When a device and its services are going to be removed from the network, the device SHOULD multicast an [ssdp:byebye](#) message corresponding to each of the [ssdp:alive](#) messages it multicast that have not already expired. Similarly, if an interface change notification is received after an announcement, the device should cancel existing advertisements. Furthermore, devices need to remember their prior IP addresses in the event that some or all of them have changed. If that is the case, new advertisements have to be sent, using the same sequence described above.

All [ssdp:byebye](#) messages SHALL be sent to the IPv6 multicast address as described in section A.4.2 “Advertisement”, and SHALL contain the OPT and NLS header fields. Otherwise, the behavior is the same as IPv4. A UDA 2.0 example of an [ssdp:byebye](#) message has the following syntax.

```
NOTIFY * HTTP/1.1
HOST: [FF02::C]:1900
NT: notification type
OPT: "http://schemas.upnp.org/upnp/1/0/"; ns=01
01-NLS: same value as BOOTID field value
NTS: ssdp:byebye
BOOTID.UPNP.ORG: number increased each time device sends an initial announce or update message
CONFIGID.UPNP.ORG: number used for caching description information
USN: composite identifier for the advertisement
```

### A.4.4 Advertisement: Device update

All [ssdp:update](#) messages SHALL be sent to the IPv6 multicast address as described in section A.4.2 “Advertisement”, and SHALL contain the OPT and NLS header fields. Otherwise, the behavior is the same as IPv4. A UDA 2.0 example of an [ssdp:update](#) message has the following syntax.

```
NOTIFY * HTTP/1.1
HOST: [FF02::C]:1900
LOCATION: URL for UPnP description for root device
NT: notification type
OPT: "http://schemas.upnp.org/upnp/1/0/"; ns=01
01-NLS: same value as BOOTID field value
NTS: ssdp:update
USN: composite identifier for the advertisement
BOOTID.UPNP.ORG: BOOTID value that the device has used in its previous announcements
CONFIGID.UPNP.ORG: number used for caching description information
NEXTBOOTID.UPNP.ORG: new BOOTID value that the device will use in subsequent announcements
SEARCHPORT.UPNP.ORG: number identifies port on which device responds to unicast M-SEARCH
```

### A.4.5 Search

When a control point is added to the network, it SHALL send multicast M-SEARCH requests on all non-temporary GUA, ULA, Link-Local, and IPv4 addresses. Aside from using an IPv6 multicast address and including an IPv6 address in the header fields, M-SEARCH messages are unchanged. An example of an M-SEARCH message has the following syntax. In addition, M-SEARCH messages MAY be unicast to IPv6 addresses of known devices, similar to IPv4 unicast M-SEARCH messages.

```
M-SEARCH * HTTP/1.1
HOST: [FF02::C]:1900
MAN: "ssdp:discover"
MX: seconds to delay response
ST: search target
```

#### A.4.6 Search response

To be found, a device SHALL send a response to the source IP address and port that sent the request to the multicast address, and SHALL include the OPT and NLS header fields in the message. It SHALL select a source address for this response according to RFC 6724. A UDA 2.0 example of a search response message has the following syntax.

```
HTTP/1.1 200 OK
CACHE-CONTROL: max-age = seconds until advertisement expires
DATE: when response was generated
EXT:
LOCATION: URL for UPnP description of this device
SERVER: OS/version UPnP/2.0 product/version
OPT: "http://schemas.upnp.org/upnp/1/0/"; ns=01
01-NLS: same value as BOOTID field value
ST: search target
BOOTID.UPNP.ORG: number increased each time device sends an initial announce or update message
CONFIGID.UPNP.ORG: number used for caching description information
USN: composite identifier for the advertisement
```

#### A.5 Description

Description documents SHALL be sent using the same address on which the HTTP GET was received. Otherwise, behavior is the same as IPv4.

#### A.6 Control

Responses to SOAP messages during the Control phase SHALL be sent on the same address on which the request was received. Otherwise, behavior is the same as IPv4.

#### A.7 Eventing

When subscribing to events over IPv6, the <deliveryURL> (or URLs) specified in the CALLBACK header field of the SUBSCRIBE message SHALL be reachable by the device. This means, for example, when sending a SUBSCRIBE request to a device using a Link-Local IPv6 address, the <deliveryURL> SHALL be the same IPv6 address used as the source address, which is a node's Link-Local address.

IPv4 addresses SHALL NOT be included in the CALLBACK header field of a SUBSCRIBE message sent over IPv6. IPv6 addresses SHALL NOT be included in the CALLBACK header field of a SUBSCRIBE message sent over IPv4.

IPv6 multicast event messages SHALL be sent to [FF0X::130]:7900 (with "X" being equal to the address scope used in advertisement). To receive IPv6 multicast event messages, control points SHALL listen to these addresses and ports.

To send a multicast event message, a publisher SHALL send a message with method NOTIFY in the following format. Values in *italics* below are placeholders for actual values. Refer to section 4.3.3 "Multicast Eventing: Event messages: NOTIFY" of this document for an explanation of the elements. All IP addresses contained in the event SHALL be IPv6 format and scoped as above. A UDA 2.0 example multicast event message has the following syntax.

```
NOTIFY * HTTP/1.1
HOST: [FF0X::130]:7900 *** note the address and the port number are different from SSDP ***
CONTENT-TYPE: text/xml; charset="utf-8"
USN: Unique Service Name for the publisher
SVCID: ServiceID from SCPD
NT: upnp:event
NTS: upnp:propchange
SEQ: monotonically increasing sequence count
```

```
LVL: event importance
BOOTID.UPNP.ORG: number increased each time device sends an initial announce or an update
message

<?xml version="1.0"?>
<e:propertyset xmlns:e="urn:schemas-upnp-org:event-1-0">
  <e:property>
    <variableName>new value</variableName>
  </e:property>
  <!-- Other variable names and values (if any) go here. -->
</e:propertyset>
```

## A.8 Presentation

Responses to HTTP GET requests for presentation pages SHALL be sent using the same address on the same interface on which the HTTP GET was received.

Presentation pages retrieved over IPv6 SHALL NOT contain IPv4 addresses. Presentation pages retrieved over IPv4 SHALL NOT contain IPv6 addresses.

It is RECOMMENDED that fully qualified URLs to resources on the device are not embedded in HTML presentation pages, but that relative URLs are used instead, so that the host portion of the embedded URLs does not need to be modified to match the address on which the GET was received.

## A.9 References

### A.9.1 Normative

BCP 0005

Address Allocation for Private Internets. Available at: <http://www.ietf.org/rfc/rfc1918.txt>

IANA

Internet Protocol Version 6 Multicast Addresses. Available at: <http://www.iana.org/assignments/ipv6-multicast-addresses/>.

RFC 3986

Format for Literal IPv6 Addresses in URLs. Available at: <http://www.ietf.org/rfc/rfc3986.txt>.

RFC 2774

HTTP Extension Framework. Available at: <http://www.ietf.org/rfc/rfc2774.txt>.

RFC 4862

IPv6 Stateless Address Autoconfiguration. Available at: <http://www.ietf.org/rfc/rfc4862.txt>.

RFC 3315

Dynamic Host Configuration Protocol for IPv6 (DHCPv6). Available at: <http://www.ietf.org/rfc/rfc3315.txt>.

RFC 6724

Default Address Selection for Internet Protocol version 6 (IPv6). Available at: <http://www.ietf.org/rfc/rfc6724.txt>.

RFC 3879

Deprecating Site-local Addresses. Available at: <http://www.ietf.org/rfc/rfc3879.txt>.

RFC 3986

Uniform Resource Identifiers (URI): Generic Syntax. Available at: <http://www.ietf.org/rfc/rfc3986.txt>.

RFC 4291

IP Version 6 Addressing Architecture. Available at: <http://www.ietf.org/rfc/rfc4291.txt>.

RFC 4861

Neighbor Discovery for IP Version 6 (IPv6). Available at: <http://www.ietf.org/rfc/rfc4861.txt>.

RFC 4193

Unique Local IPv6 Unicast Addresses. Available at: <http://www.ietf.org/rfc/rfc4193.txt>.

RFC 5952

A Recommendation for IPv6 Address Text Representation. Available at: <http://www.ietf.org/rfc/rfc5952.txt>.

RFC 3633

IPv6 Prefix Options for DHCPv6. Available at: <http://www.ietf.org/rfc/rfc3633.txt>.

RFC 4941

Privacy Extensions for Stateless Address Autoconfiguration in IPv6. Available at: <http://www.ietf.org/rfc/rfc4941.txt>.

## **A.9.2 Informative**

RFC 3493

Basic Socket Interface Extensions for IPv6. Available at: <http://www.ietf.org/rfc/rfc3493.txt>.

RFC 6540 (BCP 177)

IPv6 Support Required for all IP-capable Nodes. Available at: <http://www.ietf.org/rfc/rfc6540.txt>.

---

<sup>[1]</sup> This is the unicast address used by the device or control point in its multicast messages, i.e. the source IP address in the multicast messages, as well as the LOCATION URL for advertisement messages sent by devices.

---

## Annex B Schemas

[Informative]

### B.1 UPnP Device Schema

Below is the UPnP Device Schema for devices (see also section 2.10, “UPnP Datatype Schema”). The elements it defines are used in UPnP Device Templates; they are colored [green](#) throughout this specification. Immediately following this is a brief explanation of the XML Schema elements, attributes, and values used. The reference to XML Schema at the end of the section has further details.

UPnP 1.0 specifies that the namespace of the device schema is “urn:schemas-upnp-org:device-1-0”. UPnP 2.0 does not change that namespace, but redefines it in a backwards-compatible way by restricting the order in which elements can be sent and REQUIRING the presence of the `configId` attribute. Therefore, the schema below specifies the syntax to which a UPnP 2.0 Device Description Document has to adhere. UPnP 2.0 control points also should expect Device Description Documents from UPnP 1.0 devices that can send elements in any order, and will not have the `configId` attribute.

Link to schema [www.upnp.org/schemas/device-1-0.xsd](http://www.upnp.org/schemas/device-1-0.xsd).

```
<xsd:schema targetNamespace="urn:schemas-upnp-org:device-1-0"
  xmlns="urn:schemas-upnp-org:device-1-0"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="root" type="rootType"/>
  <xsd:complexType name="deviceType">
    <xsd:sequence>
      <xsd:element name="deviceType">
        <xsd:complexType>
          <xsd:simpleContent>
            <xsd:extension base="xsd:anyURI">
              <xsd:anyAttribute namespace="##other" processContents="lax"/>
            </xsd:extension>
          </xsd:simpleContent>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="friendlyName">
        <xsd:complexType>
          <xsd:simpleContent>
            <xsd:extension base="xsd:string">
              <xsd:anyAttribute namespace="##other" processContents="lax"/>
            </xsd:extension>
          </xsd:simpleContent>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="manufacturer">
        <xsd:complexType>
          <xsd:simpleContent>
            <xsd:extension base="xsd:string">
              <xsd:anyAttribute namespace="##other" processContents="lax"/>
            </xsd:extension>
          </xsd:simpleContent>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="manufacturerURL" minOccurs="0">
        <xsd:complexType>
          <xsd:simpleContent>
            <xsd:extension base="xsd:anyURI">
              <xsd:anyAttribute namespace="##other" processContents="lax"/>
            </xsd:extension>
          </xsd:simpleContent>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="modelDescription" minOccurs="0">
        <xsd:complexType>
          <xsd:simpleContent>
            <xsd:extension base="xsd:string">

```



```

        <xsd:anyAttribute namespace="##other" processContents="lax"/>
    </xsd:extension>
</xsd:simpleContent>
</xsd:complexType>
</xsd:element>
<xsd:element name="modelName">
    <xsd:complexType>
        <xsd:simpleContent>
            <xsd:extension base="xsd:string">
                <xsd:anyAttribute namespace="##other" processContents="lax"/>
            </xsd:extension>
        </xsd:simpleContent>
    </xsd:complexType>
</xsd:element>
<xsd:element name="modelNumber" minOccurs="0">
    <xsd:complexType>
        <xsd:simpleContent>
            <xsd:extension base="xsd:string">
                <xsd:anyAttribute namespace="##other" processContents="lax"/>
            </xsd:extension>
        </xsd:simpleContent>
    </xsd:complexType>
</xsd:element>
<xsd:element name="modelURL" minOccurs="0">
    <xsd:complexType>
        <xsd:simpleContent>
            <xsd:extension base="xsd:anyURI">
                <xsd:anyAttribute namespace="##other" processContents="lax"/>
            </xsd:extension>
        </xsd:simpleContent>
    </xsd:complexType>
</xsd:element>
<xsd:element name="serialNumber" minOccurs="0">
    <xsd:complexType>
        <xsd:simpleContent>
            <xsd:extension base="xsd:string">
                <xsd:anyAttribute namespace="##other" processContents="lax"/>
            </xsd:extension>
        </xsd:simpleContent>
    </xsd:complexType>
</xsd:element>
<xsd:element name="UDN">
    <xsd:complexType>
        <xsd:simpleContent>
            <xsd:extension base="xsd:anyURI">
                <xsd:anyAttribute namespace="##other" processContents="lax"/>
            </xsd:extension>
        </xsd:simpleContent>
    </xsd:complexType>
</xsd:element>
<xsd:element name="UPC" minOccurs="0">
    <xsd:complexType>
        <xsd:simpleContent>
            <xsd:extension base="xsd:string">
                <xsd:anyAttribute namespace="##other" processContents="lax"/>
            </xsd:extension>
        </xsd:simpleContent>
    </xsd:complexType>
</xsd:element>
<xsd:element name="iconList" minOccurs="0">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="icon" maxOccurs="unbounded">
                <xsd:complexType>
                    <xsd:sequence>
                        <xsd:element name="mimetype">
                            <xsd:complexType>
                                <xsd:simpleContent>
                                    <xsd:extension base="xsd:string">
                                        <xsd:anyAttribute namespace="##other" processContents="lax"/>
                                    </xsd:extension>
                                </xsd:simpleContent>
                            </xsd:complexType>
                        </xsd:element>
                    </xsd:sequence>
                </xsd:complexType>
            </xsd:element>
            <xsd:element name="width">

```

```

        <xsd:complexType>
          <xsd:simpleContent>
            <xsd:extension base="xsd:int">
              <xsd:anyAttribute namespace="##other" processContents="lax"/>
            </xsd:extension>
          </xsd:simpleContent>
        </xsd:complexType>
      </xsd:element>
    <xsd:element name="height">
      <xsd:complexType>
        <xsd:simpleContent>
          <xsd:extension base="xsd:int">
            <xsd:anyAttribute namespace="##other" processContents="lax"/>
          </xsd:extension>
        </xsd:simpleContent>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="depth">
      <xsd:complexType>
        <xsd:simpleContent>
          <xsd:extension base="xsd:int">
            <xsd:anyAttribute namespace="##other" processContents="lax"/>
          </xsd:extension>
        </xsd:simpleContent>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="url">
      <xsd:complexType>
        <xsd:simpleContent>
          <xsd:extension base="xsd:anyURI">
            <xsd:anyAttribute namespace="##other" processContents="lax"/>
          </xsd:extension>
        </xsd:simpleContent>
      </xsd:complexType>
    </xsd:element>
    <xsd:any namespace="##other" minOccurs="0" maxOccurs="unbounded"
      processContents="lax"/>
  </xsd:sequence>
  <xsd:anyAttribute namespace="##other" processContents="lax"/>
</xsd:complexType>
</xsd:element>
<xsd:any namespace="##other" minOccurs="0" maxOccurs="unbounded"
  processContents="lax"/>
</xsd:sequence>
<xsd:anyAttribute namespace="##other" processContents="lax"/>
</xsd:complexType>
</xsd:element>
<xsd:element name="serviceList">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="service" maxOccurs="unbounded">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="serviceType">
              <xsd:complexType>
                <xsd:simpleContent>
                  <xsd:extension base="xsd:anyURI">
                    <xsd:anyAttribute namespace="##other" processContents="lax"/>
                  </xsd:extension>
                </xsd:simpleContent>
              </xsd:complexType>
            </xsd:element>
            <xsd:element name="serviceId">
              <xsd:complexType>
                <xsd:simpleContent>
                  <xsd:extension base="xsd:anyURI">
                    <xsd:anyAttribute namespace="##other" processContents="lax"/>
                  </xsd:extension>
                </xsd:simpleContent>
              </xsd:complexType>
            </xsd:element>
            <xsd:element name="SCPDURL">
              <xsd:complexType>
                <xsd:simpleContent>
                  <xsd:extension base="xsd:anyURI">

```

```

        <xsd:anyAttribute namespace="##other" processContents="lax"/>
    </xsd:extension>
</xsd:simpleContent>
</xsd:complexType>
</xsd:element>
<xsd:element name="controlURL">
    <xsd:complexType>
        <xsd:simpleContent>
            <xsd:extension base="xsd:anyURI">
                <xsd:anyAttribute namespace="##other" processContents="lax"/>
            </xsd:extension>
        </xsd:simpleContent>
    </xsd:complexType>
</xsd:element>
<xsd:element name="eventSubURL">
    <xsd:complexType>
        <xsd:simpleContent>
            <xsd:extension base="xsd:anyURI">
                <xsd:anyAttribute namespace="##other" processContents="lax"/>
            </xsd:extension>
        </xsd:simpleContent>
    </xsd:complexType>
</xsd:element>
<xsd:any namespace="##other" minOccurs="0" maxOccurs="unbounded"
    processContents="lax"/>
</xsd:sequence>
<xsd:anyAttribute namespace="##other" processContents="lax"/>
</xsd:complexType>
</xsd:element>
<xsd:any namespace="##other" minOccurs="0" maxOccurs="unbounded"
    processContents="lax"/>
</xsd:sequence>
<xsd:anyAttribute namespace="##other" processContents="lax"/>
</xsd:complexType>
</xsd:element>
<xsd:element name="deviceList" type="deviceListType" minOccurs="0"/>
<xsd:element name="presentationURL" minOccurs="0">
    <xsd:complexType>
        <xsd:simpleContent>
            <xsd:extension base="xsd:anyURI">
                <xsd:anyAttribute namespace="##other" processContents="lax"/>
            </xsd:extension>
        </xsd:simpleContent>
    </xsd:complexType>
</xsd:element>
<xsd:any namespace="##other" minOccurs="0" maxOccurs="unbounded"
    processContents="lax"/>
</xsd:sequence>
<xsd:anyAttribute namespace="##other" processContents="lax"/>
</xsd:complexType>
<xsd:complexType name="deviceListType">
    <xsd:sequence>
        <xsd:element name="device" type="deviceType" maxOccurs="unbounded"/>
        <xsd:any namespace="##other" minOccurs="0" maxOccurs="unbounded"
            processContents="lax"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="rootType">
    <xsd:sequence>
        <xsd:element name="specVersion">
            <xsd:complexType>
                <xsd:sequence>
                    <xsd:element name="major">
                        <xsd:complexType>
                            <xsd:simpleContent>
                                <xsd:extension base="xsd:int">
                                    <xsd:anyAttribute namespace="##other" processContents="lax"/>
                                </xsd:extension>
                            </xsd:simpleContent>
                        </xsd:complexType>
                    </xsd:element>
                    <xsd:element name="minor">
                        <xsd:complexType>
                            <xsd:simpleContent>

```

```

        <xsd:extension base="xsd:int">
          <xsd:anyAttribute namespace="##other" processContents="lax"/>
        </xsd:extension>
      </xsd:simpleContent>
    </xsd:complexType>
  </xsd:element>
  <xsd:any namespace="##other" minOccurs="0" maxOccurs="unbounded"
    processContents="lax"/>
</xsd:sequence>
<xsd:anyAttribute namespace="##other" processContents="lax"/>
</xsd:complexType>
</xsd:element>
<xsd:element name="URLBase" minOccurs="0">
  <xsd:complexType>
    <xsd:simpleContent>
      <xsd:extension base="xsd:anyURI">
        <xsd:anyAttribute namespace="##other" processContents="lax"/>
      </xsd:extension>
    </xsd:simpleContent>
  </xsd:complexType>
</xsd:element>
<xsd:element name="device" type="deviceType"/>
<xsd:any namespace="##other" minOccurs="0" maxOccurs="unbounded"
  processContents="lax"/>
</xsd:sequence>
<xsd:attribute name="configId" type="xsd:int"/>
<xsd:anyAttribute namespace="##other" processContents="lax"/>
</xsd:complexType>
</xsd:schema>

```

#### <element>

Defines a new element. `name` attribute defines element name. `type` attribute defines the data type for the content of element. `minOccurs` attribute defines minimum number of times the element shall occur; default is `minOccurs="1"`; allowed elements have `minOccurs="0"`. `maxOccurs` attribute defines maximum number of times the element shall occur; default is `maxOccurs="1"`; elements that can appear one or more times have `maxOccurs="unbounded"`. For a more detailed description, see the XML Schema specification.

#### <complexType>

Defines a new data type, often containing sub-elements. For a more detail description, see the XML Schema specification.

#### <attribute>

Defines a new attribute for the purpose of declaring in which elements it MAY appear. Like any XML element, the `<attribute>` element MAY have attributes of its own. The `use` attribute within this element indicates whether the attribute shall be present; allowed attributes have `use="optional"`. For a more detail description, see the XML Schema specification.

## B.2 UPnP Service Schema

Below is the UPnP Service Schema (see also section 2.9, “UPnP Service Schema”). The elements it defines are used in UPnP Service Templates; they are colored **green** throughout this specification. Immediately following this is a brief explanation of the XML Schema elements, attributes, and values used. The reference to XML Schema at the end of the section has further details.

UPnP 1.0 specifies that the namespace of the service schema is “urn:schemas-upnp-org:service-1-0”. UPnP 2.0 does not change that namespace, but redefines it in a backwards-compatible way by restricting the order in which elements can be sent and requiring the presence of the `configId` attribute. Therefore, the schema below specifies the syntax to which a UPnP 2.0 SCPD has to adhere. UPnP 2.0 control points also should expect SCPDs from UPnP 1.0 devices that can send elements in any order, and will not have the `configId` attribute.

. Link to schema [www.upnp.org/schemas/service-1-0.xsd](http://www.upnp.org/schemas/service-1-0.xsd).

```
<xsd:schema targetNamespace="urn:schemas-upnp-org:service-1-0"
xmlns="urn:schemas-upnp-org:service-1-0"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="scpd" type="scpdType"/>
  <xsd:complexType name="directionType">
    <xsd:simpleContent>
      <xsd:extension base="xsd:string">
        <xsd:anyAttribute namespace="##other" processContents="lax"/>
      </xsd:extension>
    </xsd:simpleContent>
  </xsd:complexType>
  <xsd:complexType name="dataTypeType">
    <xsd:simpleContent>
      <xsd:extension base="xsd:string">
        <xsd:attribute name="type" type="xsd:string"/>
        <xsd:anyAttribute namespace="##other" processContents="lax"/>
      </xsd:extension>
    </xsd:simpleContent>
  </xsd:complexType>
  <xsd:complexType name="scpdType">
    <xsd:sequence>
      <xsd:element name="specVersion">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="major">
              <xsd:complexType>
                <xsd:simpleContent>
                  <xsd:extension base="xsd:int">
                    <xsd:anyAttribute namespace="##other" processContents="lax"/>
                  </xsd:extension>
                </xsd:simpleContent>
              </xsd:complexType>
            </xsd:element>
            <xsd:element name="minor">
              <xsd:complexType>
                <xsd:simpleContent>
                  <xsd:extension base="xsd:int">
                    <xsd:anyAttribute namespace="##other" processContents="lax"/>
                  </xsd:extension>
                </xsd:simpleContent>
              </xsd:complexType>
            </xsd:element>
            <xsd:any namespace="##other" minOccurs="0" maxOccurs="unbounded"
              processContents="lax"/>
          </xsd:sequence>
          <xsd:anyAttribute namespace="##other" processContents="lax"/>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="actionList" minOccurs="0">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="action" maxOccurs="unbounded">
              <xsd:complexType>
                <xsd:sequence>
                  <xsd:element name="name">
                    <xsd:complexType>
                      <xsd:simpleContent>
                        <xsd:extension base="xsd:string">
                          <xsd:anyAttribute namespace="##other" processContents="lax"/>
                        </xsd:extension>
                      </xsd:simpleContent>
                    </xsd:complexType>
                  </xsd:element>
                  <xsd:element name="argumentList" minOccurs="0">
                    <xsd:complexType>
                      <xsd:sequence>
                        <xsd:element name="argument" maxOccurs="unbounded">
                          <xsd:complexType>
                            <xsd:sequence>
                              <xsd:element name="name">
                                <xsd:complexType>
                                  <xsd:simpleContent>
                                    <xsd:extension base="xsd:string">
                                      <xsd:anyAttribute namespace="##other"

```

```
        processContents="lax"/>
      </xsd:extension>
    </xsd:simpleContent>
  </xsd:complexType>
</xsd:element>
<xsd:element name="direction">
  <xsd:complexType>
    <xsd:simpleContent>
      <xsd:restriction base="directionType">
        <xsd:enumeration value="in"/>
        <xsd:enumeration value="out"/>
      </xsd:restriction>
    </xsd:simpleContent>
  </xsd:complexType>
</xsd:element>
<xsd:element name="retval" minOccurs="0">
  <xsd:complexType>
    <xsd:complexContent>
      <xsd:restriction base="xsd:anyType">
        <xsd:anyAttribute namespace="##other"
          processContents="lax"/>
      </xsd:restriction>
    </xsd:complexContent>
  </xsd:complexType>
</xsd:element>
<xsd:element name="relatedStateVariable">
  <xsd:complexType>
    <xsd:simpleContent>
      <xsd:extension base="xsd:string">
        <xsd:anyAttribute namespace="##other"
          processContents="lax"/>
      </xsd:extension>
    </xsd:simpleContent>
  </xsd:complexType>
</xsd:element>
<xsd:any namespace="##other" minOccurs="0" maxOccurs="unbounded"
  processContents="lax"/>
</xsd:sequence>
<xsd:anyAttribute namespace="##other" processContents="lax"/>
</xsd:complexType>
</xsd:element>
<xsd:any namespace="##other" minOccurs="0" maxOccurs="unbounded"
  processContents="lax"/>
</xsd:sequence>
<xsd:anyAttribute namespace="##other" processContents="lax"/>
</xsd:complexType>
</xsd:element>
<xsd:any namespace="##other" minOccurs="0" maxOccurs="unbounded"
  processContents="lax"/>
</xsd:sequence>
<xsd:anyAttribute namespace="##other" processContents="lax"/>
</xsd:complexType>
</xsd:element>
<xsd:element name="serviceStateTable">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="stateVariable" maxOccurs="unbounded">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="name">
              <xsd:complexType>
                <xsd:simpleContent>
                  <xsd:extension base="xsd:string">
                    <xsd:anyAttribute namespace="##other"
                      processContents="lax"/>
                  </xsd:extension>
                </xsd:simpleContent>
              </xsd:complexType>
            </xsd:element>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:element name="dataType">
```

```
<xsd:complexType>
  <xsd:simpleContent>
    <xsd:restriction base="dataTypeType">
      <xsd:enumeration value="ui1"/>
      <xsd:enumeration value="ui2"/>
      <xsd:enumeration value="ui4"/>
      <xsd:enumeration value="i1"/>
      <xsd:enumeration value="i2"/>
      <xsd:enumeration value="i4"/>
      <xsd:enumeration value="int"/>
      <xsd:enumeration value="r4"/>
      <xsd:enumeration value="r8"/>
      <xsd:enumeration value="number"/>
      <xsd:enumeration value="fixed.14.4"/>
      <xsd:enumeration value="float"/>
      <xsd:enumeration value="char"/>
      <xsd:enumeration value="string"/>
      <xsd:enumeration value="date"/>
      <xsd:enumeration value="dateTime"/>
      <xsd:enumeration value="dateTime.tz"/>
      <xsd:enumeration value="time"/>
      <xsd:enumeration value="time.tz"/>
      <xsd:enumeration value="boolean"/>
      <xsd:enumeration value="bin.base64"/>
      <xsd:enumeration value="bin.hex"/>
      <xsd:enumeration value="uri"/>
      <xsd:enumeration value="uuid"/>
    </xsd:restriction>
  </xsd:simpleContent>
</xsd:complexType>
</xsd:element>
<xsd:element name="defaultValue" minOccurs="0">
  <xsd:complexType>
    <xsd:simpleContent>
      <xsd:extension base="xsd:string">
        <xsd:anyAttribute namespace="##other" processContents="lax"/>
      </xsd:extension>
    </xsd:simpleContent>
  </xsd:complexType>
</xsd:element>
<xsd:choice minOccurs="0">
  <xsd:element name="allowedValueList">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="allowedValue" maxOccurs="unbounded">
          <xsd:complexType>
            <xsd:simpleContent>
              <xsd:extension base="xsd:string">
                <xsd:anyAttribute namespace="##other" processContents="lax"/>
              </xsd:extension>
            </xsd:simpleContent>
          </xsd:complexType>
        </xsd:element>
        <xsd:any namespace="##other" minOccurs="0" maxOccurs="unbounded"
          processContents="lax"/>
      </xsd:sequence>
      <xsd:anyAttribute namespace="##other" processContents="lax"/>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="allowedValueRange">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="minimum">
          <xsd:complexType>
            <xsd:simpleContent>
              <xsd:extension base="xsd:double">
                <xsd:anyAttribute namespace="##other" processContents="lax"/>
              </xsd:extension>
            </xsd:simpleContent>
          </xsd:complexType>
        </xsd:element>
        <xsd:element name="maximum">
          <xsd:complexType>
            <xsd:simpleContent>
              <xsd:extension base="xsd:double">
```

```

        <xsd:anyAttribute namespace="##other" processContents="lax"/>
    </xsd:extension>
</xsd:simpleContent>
</xsd:complexType>
</xsd:element>
<xsd:element name="step" minOccurs="0">
    <xsd:complexType>
        <xsd:simpleContent>
            <xsd:extension base="xsd:double">
                <xsd:anyAttribute namespace="##other" processContents="lax"/>
            </xsd:extension>
        </xsd:simpleContent>
    </xsd:complexType>
</xsd:element>
<xsd:any namespace="##other" minOccurs="0" maxOccurs="unbounded"
processContents="lax"/>
</xsd:sequence>
<xsd:anyAttribute namespace="##other" processContents="lax"/>
</xsd:complexType>
</xsd:element>
</xsd:choice>
<xsd:any namespace="##other" minOccurs="0" maxOccurs="unbounded"
processContents="lax"/>
</xsd:sequence>
<xsd:attribute name="sendEvents" default="1">
    <xsd:simpleType>
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="1"/>
            <xsd:enumeration value="0"/>
        </xsd:restriction>
    </xsd:simpleType>
</xsd:attribute>
<xsd:attribute name="multicast" default="0">
    <xsd:simpleType>
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="1"/>
            <xsd:enumeration value="0"/>
        </xsd:restriction>
    </xsd:simpleType>
</xsd:attribute>
</xsd:complexType>
</xsd:element>
<xsd:any namespace="##other" minOccurs="0" maxOccurs="unbounded"
processContents="lax"/>
</xsd:sequence>
<xsd:anyAttribute namespace="##other" processContents="lax"/>
</xsd:complexType>
</xsd:element>
<xsd:any namespace="##other" minOccurs="0" maxOccurs="unbounded" processContents="lax"/>
</xsd:sequence>
<xsd:attribute name="configId" type="xsd:int"/>
<xsd:anyAttribute namespace="##other" processContents="lax"/>
</xsd:complexType>
</xsd:schema>

```

#### <element>

Defines a new element. `name` attribute defines element name. `type` attribute defines the data type for the content of element. `minOccurs` attribute defines minimum number of times the element shall occur; default is `minOccurs="1"`; allowed elements have `minOccurs="0"`. `maxOccurs` attribute defines maximum number of times the element shall occur; default is `maxOccurs="1"`; elements that can appear one or more times have `maxOccurs="unbounded"`. For a more detailed description, see the XML Schema specification.

#### <complexType>

Defines a new data type, often containing sub-elements. For a more detail description, see the XML Schema specification.

#### <attribute>

Defines a new attribute for the purpose of declaring in which elements it MAY appear. Like any XML element, the `<attribute>` element MAY have attributes of its own. The `use` attribute within this element indicates whether the attribute shall be present; allowed attributes have `use="optional"`. For a more detail description, see the XML Schema specification.



### B.3 UPnP Control Schema

Below is the template for UPnP Control Schemas (see also section 3.2.3, “UPnP Action Schema”). The elements it defines are used in actions and action responses; they are colored [green](#) throughout this specification. Immediately following this is a brief explanation of the XML Schema elements, attributes, and values used. The reference to XML Schema at the end of the section has further details.

Link to schema [www.upnp.org/schemas/service\[serviceType\].xsd](http://www.upnp.org/schemas/service[serviceType].xsd).

```
<xsd:schema targetNamespace="urn:schemas-upnp-org:service:[serviceType]:v"
xmlns="urn:schemas-upnp-org:service:[serviceType]:v"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="[actionName]" type="[actionName]Type"/>
  <xsd:element name="[actionName]Response" type="[actionName]ResponseType"/>
  <xsd:complexType name="[actionName]Type">
    <xsd:sequence>
      <!-- Use this for an argument of simple content. -->
      <xsd:element name="[argumentName]">
        <xsd:complexType>
          <xsd:simpleContent>
            <xsd:extension base="[argumentType]">
              <xsd:anyAttribute namespace="##other" processContents="lax"/>
            </xsd:extension>
          </xsd:simpleContent>
        </xsd:complexType>
      </xsd:element>
      <!-- Use this for an argument of complex content. -->
      <xsd:element name="[argumentName]" type="[argumentType]"/>
      <!-- Other arguments and their types go here, if any. -->
      <xsd:any namespace="##other" minOccurs="0" maxOccurs="unbounded" processContents="lax"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="[actionName]ResponseType">
    <xsd:sequence>
      <!-- Use this for an argument of simple content. -->
      <xsd:element name="[argumentName]">
        <xsd:complexType>
          <xsd:simpleContent>
            <xsd:extension base="[argumentType]">
              <xsd:anyAttribute namespace="##other" processContents="lax"/>
            </xsd:extension>
          </xsd:simpleContent>
        </xsd:complexType>
      </xsd:element>
      <!-- Use this for an argument of complex content. -->
      <xsd:element name="[argumentName]" type="[argumentType]"/>
      <!-- Other arguments and their types go here, if any. -->
      <xsd:any namespace="##other" minOccurs="0" maxOccurs="unbounded" processContents="lax"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

#### <element>

Defines a new element. `name` attribute defines element name. `type` attribute defines the data type for the content of element. `minOccurs` attribute defines minimum number of times the element shall occur; default is `minOccurs="1"`; allowed elements have `minOccurs="0"`. `maxOccurs` attribute defines maximum number of times the element shall occur; default is `maxOccurs="1"`; elements that can appear one or more times have `maxOccurs="unbounded"`. For a more detailed description, see the XML Schema specification.

#### <complexType>

Defines a new data type, often containing sub-elements. For a more detail description, see the XML Schema specification.

#### <attribute>

Defines a new attribute for the purpose of declaring in which elements it MAY appear. Like any XML element, the <attribute> element MAY have attributes of its own. The `use` attribute within this element indicates whether the attribute shall be present; allowed attributes have `use="optional"`. For a more detail description, see the XML Schema specification.

## B.4 UPnP Error Schema

Below is the template for UPnP Error Schemas (see also section 3.2.6, “UPnP Error Schema”). The elements it defines are used in error messages; they are colored [green](#) throughout this specification. Immediately following this is a brief explanation of the XML Schema elements, attributes, and values used. The reference to XML Schema at the end of the section has further details.

Link to schema <http://www.upnp.org/schemas/control-1-0.xsd>.

```
<xsd:schema targetNamespace="urn:schemas-upnp-org:control-1-0"
xmlns="urn:schemas-upnp-org:control-1-0"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="UPnPError" type="UPnPErrorType"/>
  <xsd:complexType name="UPnPErrorType">
    <xsd:sequence>
      <xsd:element name="errorCode">
        <xsd:complexType>
          <xsd:simpleContent>
            <xsd:extension base="xsd:int">
              <xsd:anyAttribute namespace="##other" processContents="lax"/>
            </xsd:extension>
          </xsd:simpleContent>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="errorDescription">
        <xsd:complexType>
          <xsd:simpleContent>
            <xsd:extension base="xsd:string">
              <xsd:anyAttribute namespace="##other" processContents="lax"/>
            </xsd:extension>
          </xsd:simpleContent>
        </xsd:complexType>
      </xsd:element>
      <xsd:any namespace="##other" minOccurs="0" maxOccurs="unbounded" processContents="lax"/>
    </xsd:sequence>
    <xsd:anyAttribute namespace="##other" processContents="lax"/>
  </xsd:complexType>
</xsd:schema>
```

#### <element>

Defines a new element. `name` attribute defines element name. `type` attribute defines the data type for the content of element. `minOccurs` attribute defines minimum number of times the element shall occur; default is `minOccurs="1"`; allowed elements have `minOccurs="0"`. `maxOccurs` attribute defines maximum number of times the element shall occur; default is `maxOccurs="1"`; elements that can appear one or more times have `maxOccurs="unbounded"`. For a more detailed description, see the XML Schema specification.

#### <complexType>

Defines a new data type, often containing sub-elements. For a more detail description, see the XML Schema specification.

#### <attribute>

Defines a new attribute for the purpose of declaring in which elements it MAY appear. Like any XML element, the <attribute> element MAY have attributes of its own. The `use` attribute within this element indicates whether the attribute shall be present; allowed attributes have `use="optional"`. For a more detail description, see the XML Schema specification.

## B.5 UPnP Event Schema

Below is the template for the UPnP Event Schema (see also section 4.4, “UPnP Event Schema”). The elements it defines are used in event notifications; they are colored [green](#) throughout this specification. Immediately following this is a brief explanation of the XML Schema elements, attributes, and values used. The reference to XML Schema at the end of the section has further details.

Link to schema <http://www.upnp.org/schemas/event-1-0.xsd>.

```
<xsd:schema targetNamespace="urn:schemas-upnp-org:event-1-0"
  xmlns="urn:schemas-upnp-org:event-1-0"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="propertyset" type="propertysetType"/>
  <xsd:complexType name="propertysetType">
    <xsd:sequence>
      <xsd:element name="property" maxOccurs="unbounded">
        <xsd:complexType>
          <xsd:sequence>
            <!-- Use this for a stateVariable of simple content. -->
            <xsd:element name="[stateVariableName]" form="unqualified">
              <xsd:complexType>
                <xsd:simpleContent>
                  <xsd:extension base="[stateVariableType]">
                    <xsd:anyAttribute namespace="##other" processContents="lax"/>
                  </xsd:extension>
                </xsd:simpleContent>
              </xsd:complexType>
            </xsd:element>
            <!-- Use this for a stateVariable of complex content. -->
            <xsd:element name="[stateVariableName]" type="[stateVariableType]"
              form="unqualified"/>
            <xsd:any namespace="##other" minOccurs="0" maxOccurs="unbounded"
              processContents="lax"/>
          </xsd:sequence>
          <xsd:anyAttribute namespace="##other" processContents="lax"/>
        </xsd:complexType>
      </xsd:element>
      <xsd:any namespace="##other" minOccurs="0" maxOccurs="unbounded" processContents="lax"/>
    </xsd:sequence>
    <xsd:anyAttribute namespace="##other" processContents="lax"/>
  </xsd:complexType>
</xsd:schema>
```

<element>

Defines a new element. name attribute defines element name. type attribute defines the data type for the content of element. minOccurs attribute defines minimum number of times the element shall occur; default is minOccurs="1"; allowed elements have minOccurs="0". maxOccurs attribute defines maximum number of times the element shall occur; default is maxOccurs="1"; elements that can appear one or more times have maxOccurs="unbounded". For a more detailed description, see the XML Schema specification.

<complexType>

Defines a new data type, often containing sub-elements. For a more detail description, see the XML Schema specification.

<attribute>

Defines a new attribute for the purpose of declaring in which elements it MAY appear. Like any XML element, the <attribute> element MAY have attributes of its own. The use attribute within this element indicates whether the attribute shall be present; allowed attributes have use="optional". For a more detail description, see the XML Schema specification.

## B.6 UPnP Cloud Schema

Below is the UPnP Device Schema for Cloud (see Annex C). The elements it defines are used in UPnP Cloud Templates; they are colored [blue](#) [trebuchet](#) [underlined](#) throughout this specification.

The schema below specifies the syntax to which a UPnP 2.0 Device or UPnP 2.0 control points conforms to when supporting Annex C (Cloud 1.0).

Link to schema <http://www.upnp.org/schemas/cloud-1-0.xsd>.

```
<?xml version="1.0"?>
<xsd:schema
  attributeFormDefault="unqualified"
  elementFormDefault="qualified"
  targetNamespace="urn:schemas-upnp-org:cloud-1-0"
  xmlns="urn:schemas-upnp-org:cloud-1-0"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:annotation>
    <xsd:documentation xml:lang="en">
      Date February 20, 2015
      Note that all schema supplied by the UPnP Forum Cloud Task Force are for
      informational use only and that the standardized UDA describes the
      normative requirements for these schema. Some schema provided do not
      necessarily embody requirements regarding number of element occurrences
      allowed or their ordering or specific combination.
    </xsd:documentation>
  </xsd:annotation>
  <xsd:simpleType name="hashType">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="uda"/>
      <xsd:enumeration value="sha-256"/>
    </xsd:restriction>
  </xsd:simpleType>
  <xsd:simpleType name="typeType">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="describe"/>
      <xsd:enumeration value="described"/>
      <xsd:enumeration value="error"/>
    </xsd:restriction>
  </xsd:simpleType>
  <!-- =====
  In UCA 1.0 <uc> and <query> elements are not expected to occur in
  the same UCA stanza.
  ===== -->
  <xsd:element name="uc">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="configIdCloud" minOccurs="0" maxOccurs="1">
          <xsd:complexType>
            <xsd:simpleContent>
              <xsd:extension base="xsd:string">
                <xsd:attribute name="hash" type="hashType" use="required"/>
                <!-- In UCA 1.0 the only expected value for the "hash"
                attribute is "uda" or "sha-256" -->
              </xsd:extension>
            </xsd:simpleContent>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
      <xsd:attribute type="xsd:string" name="serviceId" use="optional"/>
      <!-- In UCA 1.0 serviceId attribute and configIdCloud element are not
      expected to be used in the same <uc> element -->
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="query">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:any namespace="##other" minOccurs="0" maxOccurs="unbounded"
          processContents="lax"/>
      </xsd:sequence>
      <xsd:attribute type="xsd:string" name="type" use="required"/>
      <!-- In UCA 1.0 the only expected values for the "type"
      attribute is "describe", "described", and "error"
      and only in specific scenarios -->
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

```
<xsd:attribute type="xsd:string" name="name" use="required"/>
<!-- In UCA 1.0 the expected value for the "name"
      attribute is the USN -->
</xsd:complexType>
</xsd:element>
</xsd:schema>
```

#### <element>

Defines a new element. name attribute defines element name. type attribute defines the data type for the content of element. minOccurs attribute defines minimum number of times the element shall occur; default is minOccurs="1"; allowed elements have minOccurs="0". maxOccurs attribute defines maximum number of times the element shall occur; default is maxOccurs="1"; elements that can appear one or more times have maxOccurs="unbounded". For a more detailed description, see the XML Schema specification.

#### <complexType>

Defines a new data type, often containing sub-elements. For a more detail description, see the XML Schema specification.

#### <attribute>

Defines a new attribute for the purpose of declaring in which elements it MAY appear. Like any XML element, the <attribute> element MAY have attributes of its own. The use attribute within this element indicates whether the attribute shall be present; allowed attributes have use="optional". For a more detail description, see the XML Schema specification.

## B.7 Schema references

### XML

Extensible Markup Language. Available at: <http://www.w3.org/XML>.

### XML Schema (Part 1: Structures, Part 2: Datatypes)

Available at: <http://www.w3.org/TR/xmlschema-1>, <http://www.w3.org/TR/xmlschema-2>.

### Namespaces in XML

Available at: <http://www.w3.org/TR/REC-xml-names/>.

## Annex C Cloud

[Normative]

### C.1 Introduction

#### C.1.1 What is UPnP™ Cloud Technology (UCA)?

UPnP™ Cloud technology is an extension to the basic UPnP architecture described in 0 through 5 enabling Device to Device connectivity across the internet (aka “Cloud”). It is designed to re-use, in a managed way, as much of the existing, easy-to-use, flexible, standards-based connectivity currently associated with UPnP when connecting across the Internet. Using the open standards based Extensible Messaging and Presence Protocol (XMPP – see section C.4) the UCA binds the familiar UPnP protocols for Addressing, Discovery, Description, Eventing, and Control<sup>5</sup> to XMPP in a reliable, secure “Cloud” compatible way. This specific combination as described in this annex embodies the UPnP Cloud Annex or UCA.

UCA support is allowed. If UCA is implemented for a device or control point then the device or control point shall support all the required clauses of this annex for a UCCD and UCC-CP respectively. A UCS may also be implemented. If implemented, the UCS shall support all required clauses of a UCS in this annex.

#### C.1.2 Audience

The audience for this Annex includes UPnP device and control point vendors, UPnP Cloud infrastructure providers, and members of UPnP Forum working committees, and anyone else who has a need to understand the technical details of UPnP Cloud protocols.

This annex assumes the reader is familiar with the UDA Protocols (HTTP, SOAP, SSDP, GENA, TCP, UDP, IP, XML) as well as RESTful APIs; this annex makes no attempt to explain them. This annex also assumes most readers will be new to XMPP, and while it is not an XMPP tutorial, XMPP-related issues are addressed in detail given the centrality of XMPP to the UCA. This annex makes no assumptions about the readers' understanding of various programming or scripting languages.

#### C.1.3 In this Annex

The UPnP Cloud Annex defines both the required binding of the UDA protocols to XMPP and the required control point, device, and infrastructure components needed to implement UPnP Cloud, including Client(UCCD and UCC-CP), Server(UCS), and add on Services (see section C.7).

Components of the UPnP Device Architecture for addressing, discovery, description, eventing and control, contained herein define only extensions or equivalent replacements for those protocol components required for communication over XMPP using the UCA. All interaction and protocol descriptions associated with the traditional UPnP Device Architecture remain unchanged, when using UCA, unless specifically modified in this specification. UDA when used in a local network remains unchanged.

The text in this specification uses the following fonts to indicate the protocol components as follows:

---

<sup>5</sup> An equivalent for UDA Presentation is not provided in UCA as this is considered a potential security issue. If a device or service presentation URL is presented it should be ignore by UCCDs and UCC-CPs.

XMPP – **Red Courier**.

UCA – **Blue Trebuchet**.

UPnP Device Architecture – same fonts as in UDA 1.1.

Pure Theory of operations (TOPs) example text will be in `black courier new` with a shaded background (not necessarily gray)<sup>6</sup>.

Additionally, clarifying text is sometimes included. There are two general forms:

- 1) Extracts from relevant XMPP specification indicated by an XEP or RFC reference followed by text enclosed in a box (see below).

From XEP or RFC
-----------------

Extracted relevant text from XEP or RFC goes here.
--

- 2) Implementation warnings shown as a "caution sign symbol" with explanatory text (see below).



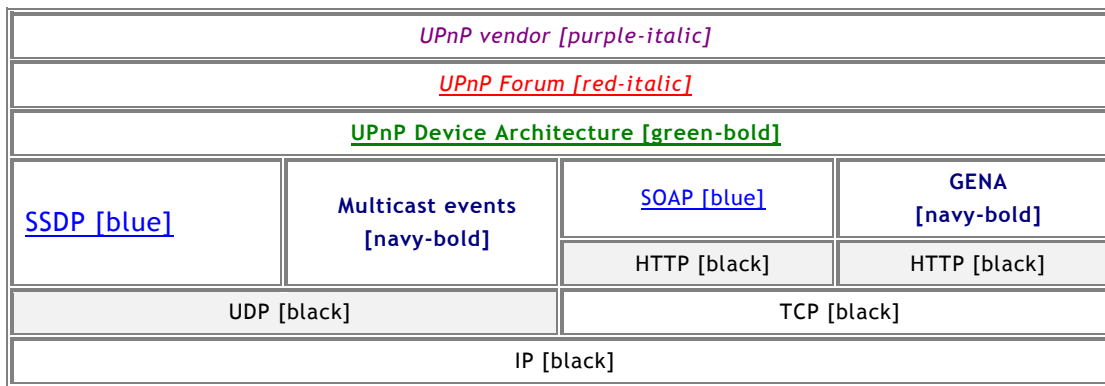
---

<sup>6</sup> Note: example text can contain whitespace and line feeds to improve readability.

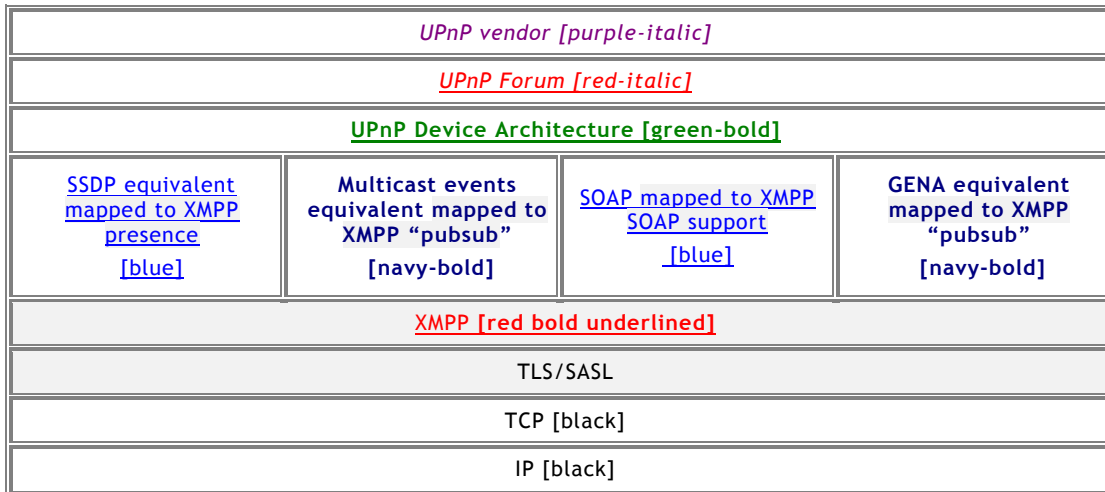
### C.1.4 UDA compared to UCA

Figure C-1 provides a high level comparison of the existing UDA protocol stack with the UCA protocol stack<sup>7</sup>. In contrast to UDA, all UCA communications are unicast, secure, and essentially reliable since it is over TCP/IP. Thus, the original UDP components are not included in the UCA. Also, the delivery method for the equivalent SOAP and GENA components are via XMPP instead of HTTP. It is possible for a device to expose one or more interfaces on both its link-local and cloud networks. Control points interacting with multiple interfaces, including both UDA and UCA compatible devices, can determine if the device(s) are the same by comparing their UDNs just as described in the multi-homed description in UDA.

**Figure C-1: — Protocol stacks UDA versus UCA<sup>8</sup>**



**UDA**



**UCA**

Figure C-2 provides a more detailed description of the UCA protocol stack. In this figure, two types of communication scenarios are illustrated. In scenario 1 (top) a UPnP Cloud Capable

<sup>7</sup> Differences are highlighted in grey.

<sup>8</sup> Note that fonts are color coded. See section "UCA Steps as Analogies to UDA" for additional information.

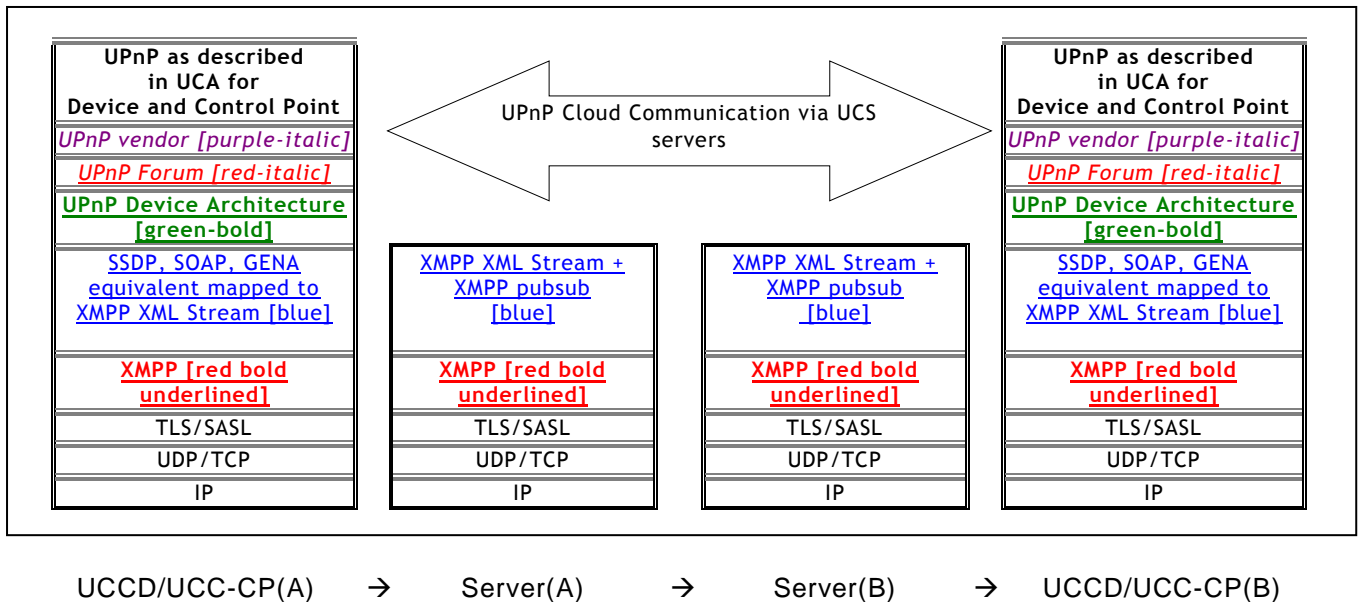


Device (aka UCCD<sup>9</sup>) or UPnP Cloud Capable Control Point (aka UCC-CP) connects through an infrastructure XMPP Server (aka UCS) to another UCCD, UCC-CP, or UCOD. This is a typical D2D communication scenario.

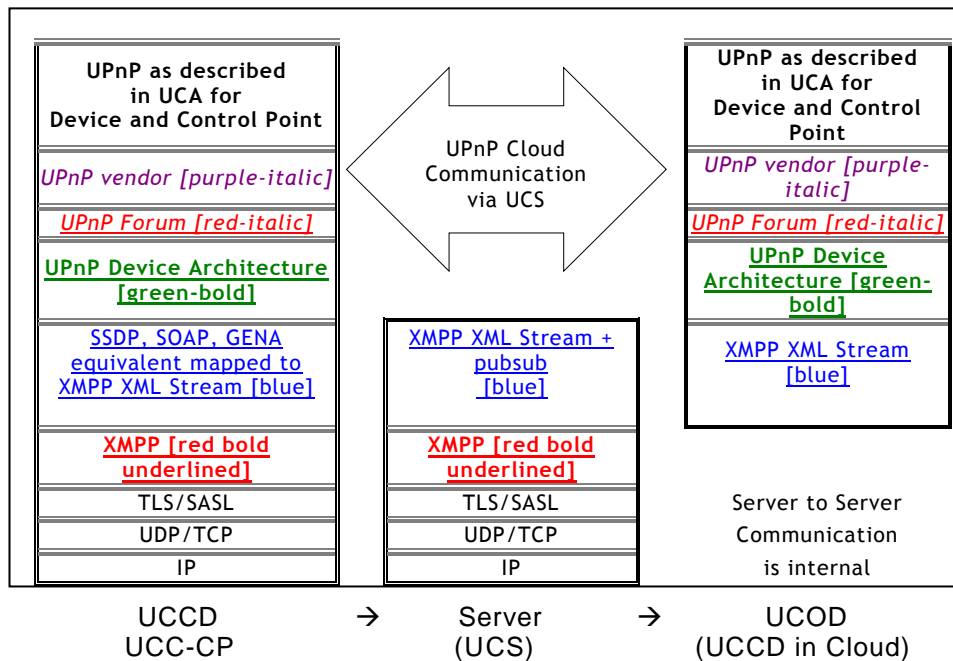
When UCCDs and UCC-CPs have different users (illustrated) the connection is typically of the type Client A (UCCD or UCC-CP) connects to server A which connects to server B which connects to Client B (UCCD or UCC-CP). When UCCDs and UCC-CPs have the same user (not illustrated) the connection will be of type Client A1 (UCCD or UCC-CP) connects to server A which connects to Client A2 (UCCD or UCC-CP), i.e. there is only one server in the path.

Another scenario (bottom) has the UCCDs and UCC-CPs connecting to devices (UCODs) and services (UCOSs) that exist only in the cloud.

**Figure C-2: — Protocol stack UCA UCCD/UCC-CP and UCA Servers (UCS or UCOD)**



<sup>9</sup> See section "Terms and Definitions" for related abbreviations.



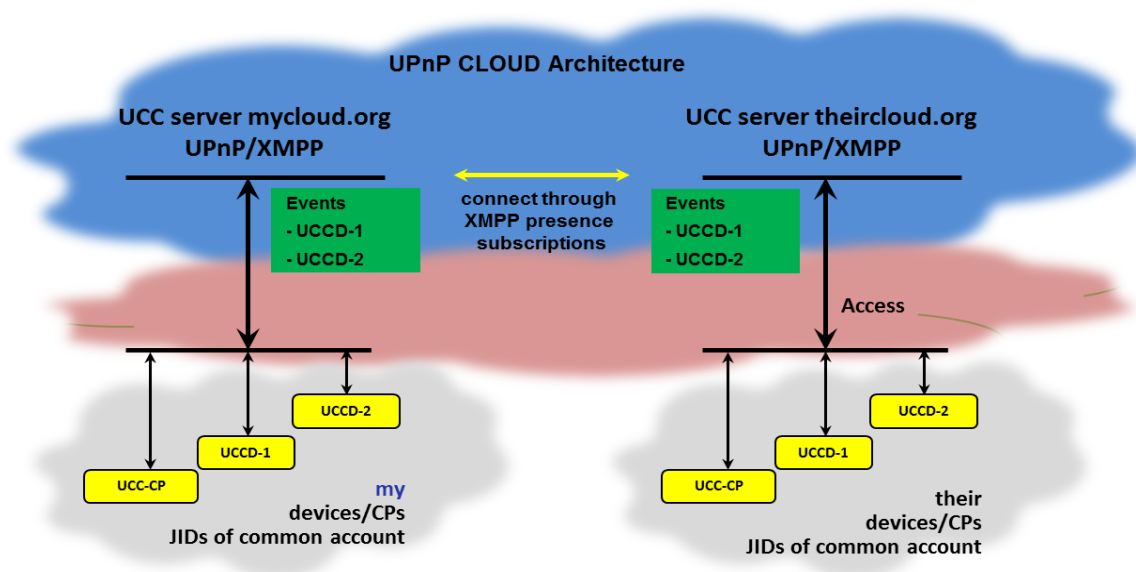
XMPP defines a technique for connecting over HTTP known as BiDirection-streams Over Synchronous HTTP (BOSH [XEP-0138]) which is targeted for browser based applications but not specifically limited to that use case. UCA support for BOSH is described in C.13.

In some cases, communications is only needed between the UCCD or UCC-CP and a UCS. Typical examples include: management of status and settings where a UCCD or UCC-CP communicates directly with its UCS; creating and publishing **PubSub** events where a UCCD or UCC-CP communicates with an add-on service discovered on its UCS; or subscribing and retrieving **PubSub** events with the UCS of a connected UCCD or UCC-CP from another user. These communication types are further defined in the next section.

### C.1.5 UCA General Communications Paths

As mentioned previously, UCA offers two paradigms for communication 1) UCCDs and UCC-CPs belonging to the same user (see term *user*) registered to the same account (see term *account*) communicating with each other and 2) UCCDs and UCC-CPs belonging to different users registered to different accounts and possibly on different servers. This is illustrated at a high level in Figure C-3 as UCCDs and UCC-CPs connected to the `mycloud.org` and `theircloud.org` UCSs respectively. This results in several communication paths between devices, control points, servers, services, and users. These specific message flows are enumerated in the next section to facilitate easier discussion in the remainder of this specification.

Figure C-3: — General UCA Configuration



### C.1.6 UCA Specific Communication Paths

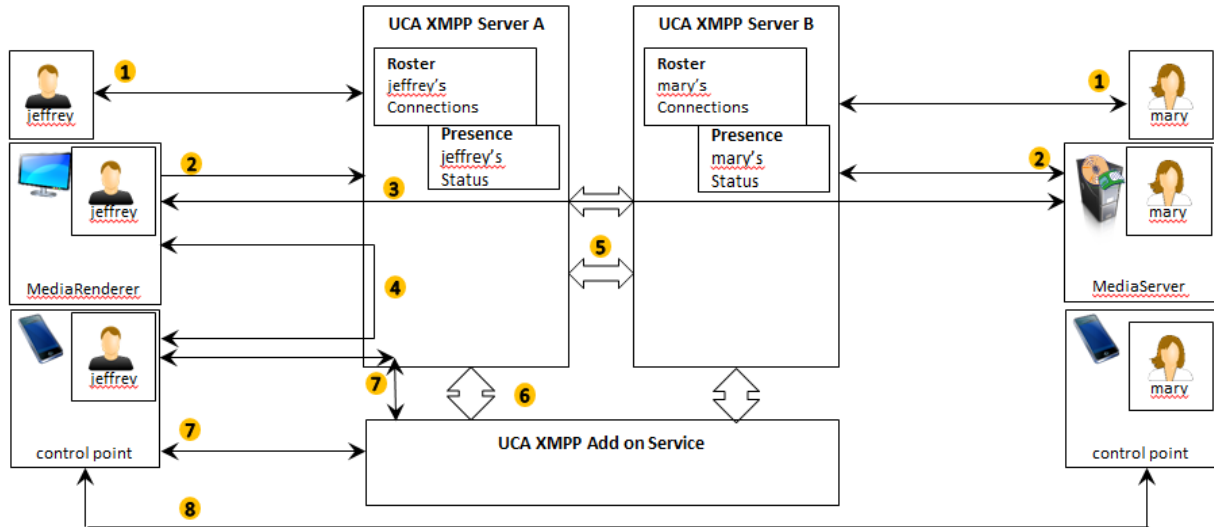
The UCA defines 8 specific types of communications 7 of which are illustrated in Figure C-4. They are:

- 1) communications between a user (with a user credential) and the UCA host server (UCS) using the defined UCA API for account management<sup>10</sup> (see section C.5.1), for example, with a browser over HTTP,
- 2) communications between a UCC-CP or UCCD (with user A credentials) with their UCA host server A using the UCA protocols,
- 3) communications between a UCC-CP or UCCD (with user A credentials) with another users connected UCC-CP or UCCD (with user B credentials) either through the same UCA host server A or through two different UCA host servers A and B using the UCA protocols,
- 4) communications between UCC-CPs and UCCDs ( with user A credentials) through their UCA host server A using the UCA protocols,
- 5) direct communications between two UCA host servers (A and B) using the XMPP protocols, this kind of communication between servers is often referred to as "federation",
- 6) communications between a user's UCA host server and required UCA add on services, such as, a [PubSub](#) service or a Multi-User Chat (MUC) conference server,
- 7) direct communications between a UCC-CP or UCCD (with user A credentials) and a server hosting a required UCA add on service, such as publishing or retrieving [PubSub](#) events.
- 8) direct communications between a UCC-CP or UCCD (with user A credentials) and a UCC-CP or UCCD (with user B credentials) for data direct data transfer as negotiated through the UCA protocols (not shown).

<sup>10</sup> Note this also includes creation of the user credential.

Figure C-4: — Specific UCA communications<sup>11</sup>

# UPnP Cloud Architecture



## C.1.7 UCA Steps as Analogies to UDA

The following section provides a high level overview of the major steps involved in establishing a UCA experience.

### Step 1: Registration

A user registers their UCC-CPs and UCCDs with a UCA host service. This is analogous to UDA Addressing. It involves several sub-steps that are described next.

- 1a) A potential UCA user finds a UCA service provider or installs a UCA server to connect their UCCDs and UCC-CPs.
- 1b) The user creates a UPnP cloud account (see section C.5.1), for example with a companion browser.
- 1c) After creating an account, the user starts a UCA cloud session between their UCC-CP or UCCD and the UCS (from step 1a) using the user credentials from (step 1b).
- 1d) The UCC-CP or UCCD autonomously creates a persistent and unique XMPP resource of general type `user@upnpcloudserver/upnp:device:resource+unique_identifier` for UCCDs or `user@upnpcloudserver/upnp:controlpoint:resource+unique_identifier` for UCC-CPs.

### Step 2: Discovery

A UCCD or UCC-CP issues XMPP `<presence>` stanzas. This is analogous to UPnP SSDP messages. It involves several sub-steps that are described next.

<sup>11</sup> Note that in this figure users "jeffrey" and "mary" are the same as general users A and B in related descriptions.

- 2a) A UCCD or UCC-CP sets its `<presence>` stanza with no `@type` attribute which is interpreted as "available" (this is equivalent to an `ssdp:alive`).
- 2b) A UCCD or UCC-CP sets its `<presence>` stanza `@type` attribute to "`unavailable`" (this is equivalent to an `ssdp:byebye`) or the UCS sends a `<presence>` stanza with `@type` attribute set to "`unavailable`" on behalf of a disconnected UCCD or UCC-CP.
- 2c) A UCCD whose `<presence>` stanza status has no `@type` attribute, which is interpreted as "available", sends a new `<presence>` stanza with a modified `configIdCloud` element (this is equivalent to an `ssdp:update`).

### Step 3 Description

A UCCD provides device type information in its name and version information sent in its `presence` communication, along with description information that can be exchanged in an `<iq>` stanza with a UCC-CP. This is analogous to UPnP Description. It involves several sub-steps that are described next.

- 3a) A UCCD sends a conforming version and hash (`configIdCloud`) of its description in its `<presence>` stanza.
- 3b) A UCC-CP caches the version and hash and if it does not have a valid description of the requests it from the UCCD using an `<iq>` stanza.
- 3b) The UCCD and UCC-CP complete a follow-up exchange in which the DDD and SCPDs are exchanged using `<iq>` stanzas.

### Step 4 Eventing

A UCCD creates and manages a UCA `PubSub collection`, if it has events, and a UCC-CP subscribes to `nodes` in that `collection`. This is analogous to UDA Eventing. It involves several sub-steps that are described next.

- 4a) A UCCD creates a conforming `PubSub collection` for UCCD events at the `PubSub` service associated with its UCA server.
- 4b) A UCC-CP issues conforming `PubSub "subscribe"` and `"unsubscribe"` queries to a specific UCCD `PubSub node` (event).
- 4c) A UCCD or UCA server issues conforming responses to the UCC-CP acknowledging the subscription request usually a `"subscribed"` or `"unsubscribed"` message.
- 4d) A UCCD sends any event to the `PubSub` service when an event occurs using the required XMPP message.

### Step 5 Control

A UCC-CP invokes an action on a UCCD by exchanging SOAP messages using the XMPP binding for SOAP. This is analogous to UDA Control. It involves several sub-steps that are described next.

- 5a) A UCC-CP invokes a SOAP based action on a UCCD using an `<iq>` stanza,
- 5b) The UCCD sends a proper SOAP response or error message to the UCCD using an `<iq>` response stanza.
- 5c) The response can include elements filtered to reflect additional negotiated and secured transport for direct UCCD to UCCD or direct UCC-CP to UCCD media or data exchanges where required or efficient.

## C.2 Terms and Definitions

### C.2.1 Acronyms

**Table C-1: — Acronyms**

Acronym	Description
BOSH	Bidirectional-streams Over Synchronous HTTP
D2D	Device-to-Device
FQDN	Fully Qualified Domain Name
MUC	Multi-User Chat
UCA	UPnP Cloud Annex
UCBS	UPnP Cloud Based Service
UCC	UPnP Cloud Capable
UCC-CP	UPnP Cloud Capable Control Point
UCCD	UPnP Cloud Capable Device
UCCD-M	UPnP Cloud Capable Device [Mobile]
UCOS	UPnP Cloud Only Service
UCS	UPnP Cloud Server
UHOD	UPnP Home Only Device
XMPP	Extensible Messaging and Presence Protocol

### C.2.2 General Cloud Terms and Definitions

Cloud, in the context of UPnP, is the logical domain, not in the user's home(s), where their UCODs, UCCDs and UCC-CPs connect, their UCBSs execute, and their cloud based content resides.

Domain is a scoped access to a subset of all available cloud devices, services and users.

Account is a domain in the cloud consisting of a username and a credential. Also, the account can contain ancillary information such as address, telephone number, email information and possibly transactional information such as credit card information. An extended concept of an account is the combination of devices, services and users registered or interacting with the account.

User, in the context of UPnP cloud, is a uniquely identifiable participant that interacts with the UPnP cloud ecosystem. A user can be an account owner or participant and can be associated with multiple cloud accounts.

Login is an identification process that allows a specific user to access their cloud account by confirming their username and credential. Login can also refer to the act of starting an active session with the cloud account. The login can be automated once initial login succeeds. Some minimal behavior equivalent to UPnP Public Role [DP] could be identified.

Owner is a user that has management rights over an account (or group) and the devices, services, and users allowed access within that account.

Home, in the context of UPnP cloud, is the logical network(s) or LAN(s) where a user's UHOD and CPDevs are connected.

UPnP Cloud Capable (UCC) means that the device or control point (CP) is capable of interaction with UPnP Cloud Based Services.

Invitation is the initiation part of registering a device, service, or user to a cloud account or cloud group.

Invited User is a user from a cloud account that has been invited to have access to devices and services in a different cloud account. The devices and services access can be granted with a per-device, per-group, or per-service granularity.

### **C.2.3 Device and Control Point Terms and Definitions**

UPnP Cloud Capable Control Point (UCC-CP) is a control point (CP) that can interact with UCCDs, UCODs, and UCOSs directly. Note that a UCC-CP is not a UCCD.

UPnP Cloud Capable Device [Mobile] (UCCD-[M]) is a non-virtual UPnP device that can interact directly with other UCCDs and UCODs via the cloud, when in the home it can interact with legacy devices over a home network. The term UCCD-M can be used to indicate that the UCCD has mobility (cell phone, tablet) that is, it will frequently get IP connectivity from outside of the home network as opposed to a UCCD that is a 100 inch Smart TV which can connect directly to the internet but is likely to stay in the same LAN. Note that a UCCD-M is also a UCCD.

UCOD (UPnP Cloud Only Device) is a UPnP device that resides only in the cloud, more or less a virtual device. Note that a UCOD is not a UCCD. UHOD (UPnP Home Only Device) is a device that only works in a home (LAN) network, that is a “legacy” device [UDA1.0], [UDA1.1], an example is MediaServer:1. It can be added to a cloud account via a CPDev

Invited Device is a device from a cloud account that has received an invitation to be registered to a different cloud account.

### **C.2.4 Service Terms and Definitions**

UPnP Cloud Based Service (UCBS) are UPnP services designed to support the UPnP cloud ecosystem. This is a broader category than a UCOD and UCOS and encompasses all cloud related components.

UPnP Cloud Only Service (UCOS) is a service that can reside only in the cloud and might only be compatible with UHODs via a CPDev.

Cloud Service is a service originating from a cloud account that is not part of a physical Device.

Service Provider (SP) is a cloud based entity that provides cloud accounts or UCOS(s) possibly including content.

Service Provider Cloud Service (SPCS) is a UCOS originating from a Service Provider available to a cloud account.

Invited Cloud Service is a cloud service from a cloud account or service provider (SP) that has received an invitation to be registered to a different cloud account.

Published Service is a service originating from a device from a home network advertised to a cloud account.

### **C.2.5 Groups**

Cloud Group is a default set of UPnP devices connected to a cloud account, often initially from the same LAN. This group can contain UHODs, UCODs, UCOSs, UCCDs and UCC-CPs. A device can belong to more than 1 cloud group.

Cloud Only Group is a default set of UPnP devices connected to a cloud account. This group can contain UCODs, UCOSs, UCCDs and UCC-CPs. A device can belong to more than 1 cloud only group.

Custom Cloud Group is a configured and managed set of UCCDs, UCODs, UHODs, UCC-CPs and users for the purpose of managing interaction with and registration to other cloud accounts and providing limited subsets of device, service, and user interactions.

Home Only Group is a default set of UPnP devices connected to a cloud account. This group can contain only UHODs.

Invited Group is a cloud, custom, or home group that has received an invitation to register to a different cloud group.

### C.3 References

- [DP] - UPnP Device Protection:1 Service. Available at <http://upnp.org/specs/gw/UPnP-gw-DeviceProtection-v1-Service.pdf>
- [RFC-6120] - Extensible Messaging and Presence Protocol XMPP: Core. Available at <http://tools.ietf.org/html/rfc6120>
- [RFC-6121] - Extensible Messaging and Presence Protocol XMPP: Instance Messaging and Presence. Available at <http://tools.ietf.org/html/rfc6121>
- [RFC-6122] - Extensible Messaging and Presence Protocol XMPP: Address Format. Available at <http://tools.ietf.org/html/rfc6122>
- [RFC-6455] – The WebSocket Protocol. Available at <http://tools.ietf.org/html/rfc6455>
- [XEP-0030] - Service Discovery, XMPP Standards Foundation, 1999-2015. Available at <http://xmpp.org/extensions/xep-0030.html>
- [XEP-0060] - Publish-Subscribe, XMPP Standards Foundation, 1999-2015. Available at <http://xmpp.org/extensions/xep-0060.html>
- [XEP-0072] - SOAP over XMPP, XMPP Standards Foundation, 1999-2015. Available at <http://xmpp.org/extensions/xep-0072.html>
- [XEP-0084] - User Avatar, XMPP Standards Foundation, 1999-2015. Available at <http://xmpp.org/extensions/xep-0084.html>
- [XEP-0124] - BiDirectional-streams Over Synchronous HTTP, 1999-2015. Available at <http://xmpp.org/extensions/xep-0124.html>
- [XEP-0133] - Service Administration, XMPP Standards Foundation, 1999-2015. Available at <http://xmpp.org/extensions/xep-0133.html>
- [XEP-0138] - Stream Compression, XMPP Standards Foundation, 1999-2015, Available at <http://xmpp.org/extensions/xep-0138.html>
- [XEP-0206] - XMPP over BOSH, XMPP Standards Foundation, 1999-2015, Available at <http://xmpp.org/extensions/xep-0206.html>
- [XEP-0248] - Collection Nodes, XMPP Standards Foundation, 1999-2015. Available at <http://xmpp.org/extensions/xep-0248.html>



[XEP-0332] - HTTP over XMPP, XMPP Standards Foundation, 1999-2015, Available at <http://xmpp.org/extensions/xep-0332.html>

## C.4 General XMPP Features

This section provides an overview of basic XMPP. The full XMPP specifications and descriptions can be found at <http://xmpp.org>.

### C.4.1 XMPP Jabber IDs or JIDs

An XMPP Jabber (historical) ID or **JID** is composed of three parts the **localpart** (analog to user name), the **domainpart** (analog to server name) and the **resourcepart** (in the UPnP case, a device or control point name). Together they form a **full JID** or the "Cloud" address of a UCCD or UCC-CP. In XMPP communication can occur between different levels of addressing: server to server, client to server, and client to client.

When UCA server to server communication occurs it is typically sent as an XMPP **domainpart** to **domainpart** and, in the case, of this specification, will appear as `mycloud.org` to `theircloud.org`. This type of communication is generally out of scope for the UCA.

When UCA client (UCCD or UCC-CP) to server communication occurs it is typically<sup>12</sup> sent as an XMPP **localpart@domainpart/resourcepart** to a **localpart@domainpart**.

However, in many cases, especially presence, UCA client communication "**to**" and "**from**" is added by the Server or UCS.

```
UCCD:jeffrey@mycloud.org/urn:schemas-upnp-org:device:MediaServer:4:uuid:abcd-1234
<presence/>
```

```
UCS:mycloud.org (to available resource ...abcd-wxyz)
```

```
<presence
  to="jeffrey@mycloud.org/urn:schemas-upnp-org:device:
    MediaServer:4:uuid:abcd-wxyz"
  from="jeffrey@mycloud.org/urn:schemas-upnp-org:device:
    MediaServer:4:uuid:abcd-1234"/>
```

Note that in this example the user or **localpart** is `jeffrey`.

Communication between a UCCD (UCC-CP) and its server is somewhat analogous to communication between a device (control point) and its local network.

When UCA client to client communication occurs it will be between **full JIDs** as **localpart@domainpart/resourcepart** to a **localpart@domainpart/resourcepart** but relayed through the client's (UCCD or UCC-CP) server first.

- UCCDs and UCC-CPs belonging to user A on Server A and connecting to user A's other UCCDs or UCC-CPs will send UCA communication using a Client A1 to Server A to Client A1 model.
- UCCDs and UCC-CPs belonging to user A1 on Server A and connecting to user A2 on server A UCCDs or UCC-CPs will send UCA communication using a Client A1 to Server A to Client A2 model.
- UCCDs and UCC-CPs belonging to user A1 on Server A and connecting to user B1 on server B UCCDs or UCC-CPs will send UCA communication using a Client A1 to Server A to Server B to Client B model.

<sup>12</sup> Specific cases in this annex will appear as `jeffrey@mycloud.org/urn:schemas-upnp-org:device:MediaServer:4:uuid:e70e9d0e-d9eb-4748-b163-636a323e7950` to `jeffrey@mycloud.org` or similar.

Note that not all client to client communication is expected to go through the servers; high bandwidth transactions will be negotiated through XMPP communication and handled in more efficient ways (see section C.9).

#### C.4.1.1 XMPP stanzas

There are three stanza types defined for XMPP: `<iq>`, `<presence>`, and `<message>`. In general, `<presence>` is broadcast between a XMPP client to an XMPP server for additional caching or distribution. A response is not expected (although the server may generate additional stanzas). An `<iq>` stanza always results in a response although it may be an error response. Typically this stanza is also client to server, or server to server. A `<message>` stanza is typically sent client to client and usually does not require a response, but one can be defined if necessary.

#### C.4.1.2 `<iq>` stanzas

XMPP `<iq>` stanzas have the following defined `@type` attribute allowed values: `"get"`, `"set"`, `"result"`, and `"error"`. They are primarily intended for request/response interactions. An `<iq>` stanza can contain only one payload and (for errors) an error child, that is errors might contain error plus original payload (see [RFC-6121], section 8.2.3).

#### C.4.1.3 `<presence>` stanzas

XMPP `<presence>` stanzas have the following defined `@type` attribute allowed values which are related to UCA: `"unavailable"` and no `@type` attribute present, which is interpreted as "available". They are used primarily as an online ("available"), off-line ("unavailable") switch which is sent to all subscribers<sup>13</sup> connected entities or in the UPnP case UCCDs and UCC-CPs.

#### C.4.1.4 `<message>` stanzas

XMPP `<message>` stanzas have the following defined `@type` attribute allowed values: `"normal"`, `"chat"`, `"groupchat"`, `"headline"`, and `"error"`. The `<message>` stanza is the primary push method in XMPP.

Embedding of other XML namespace elements in stanza payloads is supported, thus UPnP will define its own payloads as needed.

## C.5 Creating a Device or Control Point Resource

### C.5.1 Finding a UCS

Before a UCC-CP or UCCD can be connected using UCA, a UCS must be discovered and a user account created. UPnP describes the following methods for discovering UCA connective services.

Known publicly accessible UCS, UPnP Forum provides links to UCSs that have passed UCA UCS certification and that are publicly accessible. This list is maintained at hyperlink "<http://cloud.upnp.org/ucs/ver/x.x>" where x.x indicates the UCA server version the UCS is certified to. Valid versions for this specification are "1.0".

It is expected that versions above 1.0 will be released. These will be enumerated at the URI <http://cloud.upnp.org/ucs/> as hyperlinks of the form "x.x", for example:

- [1.0](#)
- [1.1](#)
- [2.0](#)

---

<sup>13</sup> A subscriber UCCD or UCC-CP is either a `resource` of the same user account or an external UCCD or UCC-CP `subscribed` to another user's UCCD or UCC-CP `presence`.

URIs at the next level "<http://cloud.upnp.org/ucs/ver>" would contain hyperlinks to the UCSs accessible<sup>14</sup> for each version, for example:

link <http://cloud.upnp.org/ucs/ver/1.0> contains 2 hyperlinks

<http://mycloud.org>

<http://theircloud.org>

link <http://cloud.upnp.org/ucs/ver/1.1> contains 2 hyperlinks

<http://mycloud.org>

link <http://cloud.upnp.org/ucs/ver/2.0> contains 1 hyperlink

<http://theircloud.org>

Vendor defined. One or more UCSs are hardcoded into the UCC-CP or UCCD. Upon initial startup the user is provided an interface to create an account. After initial account creation and login the UCC-CP or UCCD may connect automatically.

Add on services. UPnP Forum may define DCPs that allow a locally discoverable infrastructure device to advertise a service that provides discovery of a UCS.

### C.5.2 Account Creation

In general, it is expected that most UCCDs and UCC-CPs will have some form of UI supporting account registration; however this could also be done through a web interface to the supporting UCS using BOSH [XEP-0124].

[XEP-0133] describes best practices for server to server and server to component account management. For basic account setup, update, deletion, and exchange the following user form variables (var) are defined:

- accountjid,
- password,
- password-verify
- email,
- given name,
- surname

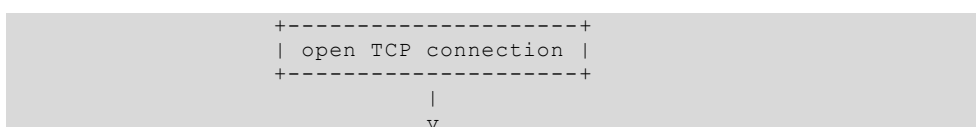
It is highly recommended that any account creation-management API, at a minimum, provision for the input, storage, and protection of the above information.

### C.5.3 Authentication

For each UCCD or UCC-CP that a user will connect to their cloud account, they will have to login to a server supporting the UCA server profile specified here. When initial login occurs, the UCCD or UCC-CP uses the bare JID or localpart@domainpart (for this example `jeffrey@mycloud.org`) and completes the preconditions (stream establishment, security negotiation (TLS, SASL) described in [RFC-6120], also known as XMPP-CORE. This process is illustrated in Figure C-5.

Note that this section is a composite of XMPP specifications [RFC-6120], [RFC-6121], [RFC-6122].

**Figure C-5: — XMPP Authentication Negotiation**



<sup>14</sup> Note that UPnP Forum does not guarantee the availability of UCS servers.



```
xml:lang="en"
xmlns="jabber:client"
xmlns:stream="http://etherx.jabber.org/streams">
<stream:features>
  <mechanisms xmlns="urn:ietf:params:xml:ns:xmpp-sasl">
    <mechanism>SCRAM-SHA-1</mechanism>
    <mechanism>PLAIN</mechanism>
  </mechanisms>
  <auth xmlns="http://jabber.org/features/iq-auth"/>
  <register xmlns="http://jabber.org/features/iq-register"/>
</stream:features>
```

C:jeffrey@mycloud.org

```
<auth
  mechanism="SCRAM-SHA-1"
  xmlns="urn:ietf:params:xml:ns:xmpp-sasl"/>
```

UCS:mycloud.org

```
<challenge xmlns="urn:ietf:params:xml:ns:xmpp-sasl">
biwsbjlpYmNkZW1vLHI9TGpvR2VSS0psZmNvU1RBS0VQVEV ...
</challenge>
```

C:jeffrey@mycloud.org

```
<response
  xmlns="urn:ietf:params:xml:ns:xmpp-sasl">
YzliaXdzLHI9TGpvR2VSS0psZmNvU1RBS0VQVEVNTndkMFBvK0FoZ0dPWkpiNWZZYj ...
</response>
```

UCS:mycloud.org

```
<success xmlns="urn:ietf:params:xml:ns:xmpp-sasl">
dj1qWGJ2VUZuUFJPRU ...
</success>
```

See [RFC-6120] section 6.4.5 describing the client server behaviour regarding SASL failure. Communication is now secure and a second secured XMPP stream or **<stream:stream>** is now started.

Note that there are actually two new streams started, one client to server and one server to C thus the new **<xml>** tag and a new stream **id** generated and sent by the Server. **Green highlighted text** indicates post SASL secured exchange. Section C.12 describes some details on how to close a XMPP session.

C:jeffrey@mycloud.org

```
<stream:stream
  to="mycloud.org"
  version="1.0"
  xml:lang="en"
  xmlns="jabber:client"
  xmlns:stream="http://etherx.jabber.org/streams">
```

UCS:mycloud.org

```
<stream:stream
  from="mycloud.org"
  to="jeffrey@mycloud.org"
  id="gPybzaOzBmaADgxKXu9UC1bprp0="
  version="1.0"
  xml:lang="en"
  xmlns="jabber:client">
  xmlns:stream="http://etherx.jabber.org/streams">
```

UCS:mycloud.org

```
<stream:features>
  <bind xmlns="urn:ietf:params:xml:ns:xmpp-bind"/>
  <session xmlns="urn:ietf:params:xml:ns:xmpp-session"/>
```

```
</stream:features>
```

Note that stream errors are unrecoverable, see [RFC-6120] section 4.9.1.

### C.5.4 Binding Devices and Control Points as a Resource

Once the secure stream has been negotiated for the UCCD or UCC-CP with the UPnP cloud server (UCS) it will send an XMPP `<bind>` notification as indicated in the example above where:

If the binding resource is a UCCD it shall bind with a `resourcepart` as the ordered concatenation of the value of the `<deviceType>` and the value of the `<UDN>` separated by a ":" either

```
resourcepart = urn:schemas-upnp-org:device:deviceType:ver:uuid:device-UUID
```

or

```
resourcepart = urn:domain-name:device:deviceType:ver:uuid:device-UUID
```

- If the binding resource is a UCC-CP it shall bind with a `resourcepart` as either:
  - the ordered concatenation of the value of urn:schemas-upnp-org:cloud-1-0:ControlPoint:ver: and either the value of control point `<ID>` as defined in DeviceProtection Service [DP] if DeviceProtection Service is implemented , that is,

```
resourcepart = urn:schemas-upnp-org:cloud-1-0:ControlPoint:ver:<ID>
```

or

- a UUID value meeting the same requirements as the `device-UUID` that is,

```
resourcepart = urn:schemas-upnp-org:cloud-1-0:ControlPoint:ver:uuid
```

For UCA 1.0, the value of the control point version `ver` is "1".

Continuing the example from above, upon receiving the bind notification, the UCCD "`MediaServer:4`" will now bind as:

```
jeffrey@mycloud.org/urn:schemas-upnp-org:device-1-1:MediaServer:4:uuid:e70e9d0e-d9eb-4748-b163-636a323e7950
```

by sending an `<iq>` bind request as follows:

```
C:jeffrey@mycloud.org/urn:schemas-upnp-org:device:MediaServer:4:uuid:e70e9d0e-d9eb-4748-b163-636a323e7950
```

```
<iq
  to="mycloud.org"
  from="jeffrey@mycloud.org"
  type="set"
  id="yhcl3a95"
  <bind xmlns="urn:ietf:params:xml:ns:xmpp-bind"/>
    <resource>
      urn:schemas-upnp-org:device:MediaServer:4:
        uuid:e70e9d0e-d9eb-4748-b163-636a323e7950
    </resource>
  </bind>
</iq>
```

Note, that the value of the `iq@id` attribute is determined by the endpoint (usually a UCS)<sup>15</sup>. A UCS shall accept a conforming request as a new resource with the response:

UCS:mycloud.org

```
<iq
  to="jeffrey@mycloud.org
  from="mycloud.org"
  type="result"
  id="yhcl3a95"
  <bind xmlns="urn:ietf:params:xml:ns:xmpp-bind"/>
    <jid>jeffrey@mycloud.org/urn:schemas-upnp-org:device:
      MediaServer:4:uuid:e70e9d0e-d9eb-4748-b163-636a323e7950</jid>
  </bind>
</iq>
```

Error messages received prior to resource binding shall conform to standard XMPP messages as defined in [RFC-6120] and are not considered a UCA error.

Note that a properly implemented UCCD or UCC-CP should not create a resource uniqueness conflict as described in section 7.7.2.2 [RFC-6120], that is uniqueness is guaranteed between all UCCD and UCC-CP resources. A UCS should not attempt to change the name of a requested resource, if it does, the UCCD or UCC-CP requesting the resource bind shall disconnect the stream and try to reconnect according to standard XMPP procedures. A UCCD or UCC-CP shall not accept an XMPP bind that does not conform to its advertised as described above..

Upon a successful bind, the MediaServer:4 UCCD is now connected with its own globally unique full JID. This is the UCCD cloud address. Note, a UCCD is allowed to have multiple cloud addresses.

UCCD:jeffrey@mycloud.org/urn:schemas-upnp-  
org:device:MediaServer:4:uuid:e70e9d0e-d9eb-4748-b163-636a323e7950

```
<iq
  id="yhc95a13"
  from="jeffrey@mycloud.org/urn:schemas-upnp-org:device:MediaServer:4:
    uuid:e70e9d0e-d9eb-4748-b163-636a323e7950"
  to="jeffrey@mycloud.org"
  type="set"
  <query xmlns="jabber:iq:roster">
    <item jid="jeffrey@mycloud.org/urn:schemas-upnp-org:device:
      MediaServer:4:uuid:e70e9d0e-d9eb-4748-b163-636a323e7950"
      name="jeffrey MediaServer 4 on cell"/>
  </query>
</iq>
```

UCS:mycloud.org

```
<iq
  id="yhc95a13"
  to="jeffrey@mycloud.org/urn:schemas-upnp-org:device:
    MediaServer:4:uuid:e70e9d0e-d9eb-4748-b163-636a323e7950"
  from="mycloud.org"
  type="result">
  <query xmlns="jabber:iq:roster" ver="ver1">
    <item jid="jeffrey@mycloud.org/urn:schemas-upnp-org:device:
      MediaServer:4:uuid:e70e9d0e-d9eb-4748-b163-636a323e7950"
      name="jeffrey MediaServer 4 on cell"/>
  </query>
</iq>
```

---

<sup>15</sup> id values in the examples are not randomized or sequenced, as is common practice, for illustration purposes only.

#### C.5.4.1 Limitations on stanza redirection.

In some cases, XMPP allows for the redirection of stanzas to an alternative "available" resource (see [RFC-6120] section 10.5.4) when a full JID matching the "to" attribute is not "available". For example, a message of type 4 intended for `jeffrey`'s MediaRender (which is "unavailable") could be redirected to jeffrey's MediaServer (which is "available"). For UCA communications this is prohibited as follows:

UCCDs and UCC-CPs shall ignore any received stanza that does not have a "from" attribute with a value compliant to a UCA resource as defined above with the following exceptions:

- The stanza is a response to a UCCD or UCC-CP request to the UCS or affiliated service (such as PubSub). Note that the UCCD or UCC-CP should check the stanza @id attribute to confirm.
- The stanza is a notification from the UCS or affiliated service.
- The stanza is a message specifically identified in this specification indicating the allowed override of the prohibition.

A UCS should refrain from redirecting valid UCA messages sent to an "unavailable" UCA resource to an "available" UCA resource and instead send appropriate error messages as described in [RFC-6120].

Note that UCCDs and UCC-CPs are allowed to be combined with other applications that share a common XMPP interface, such as "chat". However, non-UCA stanzas should, instead be sent to the bare JID and resource priority used for any additional routing. Figure C-6 illustrates these two scenarios. In case 1, a MediaServer sends a UCA stanza to an "unavailable" MediaRender. Instead of delivering the stanza to the "available" MediaServer or UCC-CP the appropriate error is returned. In case 2, a non-UCA stanza addressed to the bare JID is sent and delivered to the "available" resource with the highest priority, in this case Jeffrey's UCC-CP; in other terms, UCCDs and UCC-CPs should ignore stanzas of type="chat" unless a chat interface is provided.

#### C.5.4.2 Handling resource priority

The use of priority in UCCDs and UCC-CPs is allowed in UCA, however its behavior is not guaranteed since XMPP allows a server to override requested priority as described below.

From [RFC-6121] section 4.7.2.3

The client's server MAY override the priority value provided by the client (e.g. in order to impose a message handling rule of delivering a message intended for the account's bare JID to all of the account's available resources). If the server does so it MUST communicate the modified priority value when it echoes the client's presence back to itself and sends the presence notification to the user's contacts (because this modified priority value is typically the default value of zero communicating the modified priority value can be done by not including the <priority/> child element).

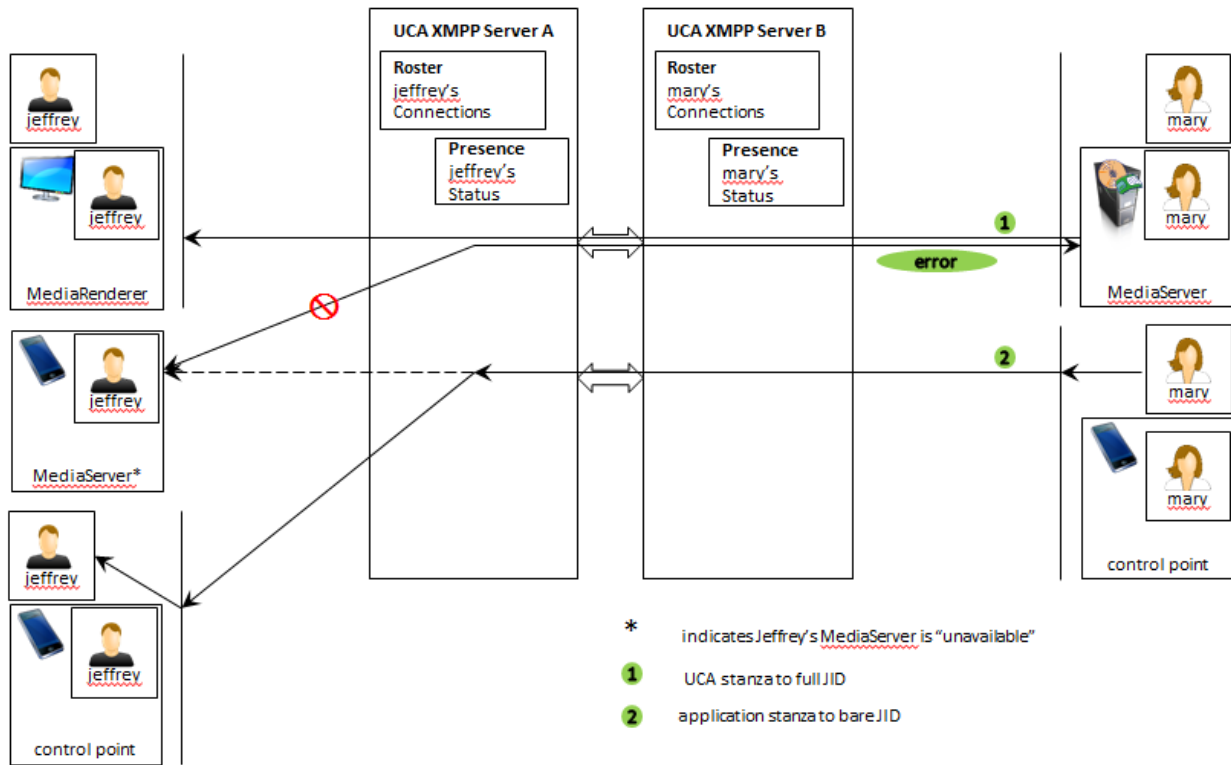
However, it is suggested that UCS recognize the following recommended priorities for UCCDs and UCC-CPs:

- UCCDs with no additional XMPP features: priority range of [-100 to -33].
- UCCDs with additional XMPP features: priority range of [ 1 to 66].
- UCC-CPs with no additional XMPP features: priority range of [ -66 to -1].
- UCC-CPS with additional XMPP feature: priority range of [ 33 to 100].

**Figure C-6: — Stanza routing for applications with UCA and other XMPP functionality.**



# UCA Stanza Routing

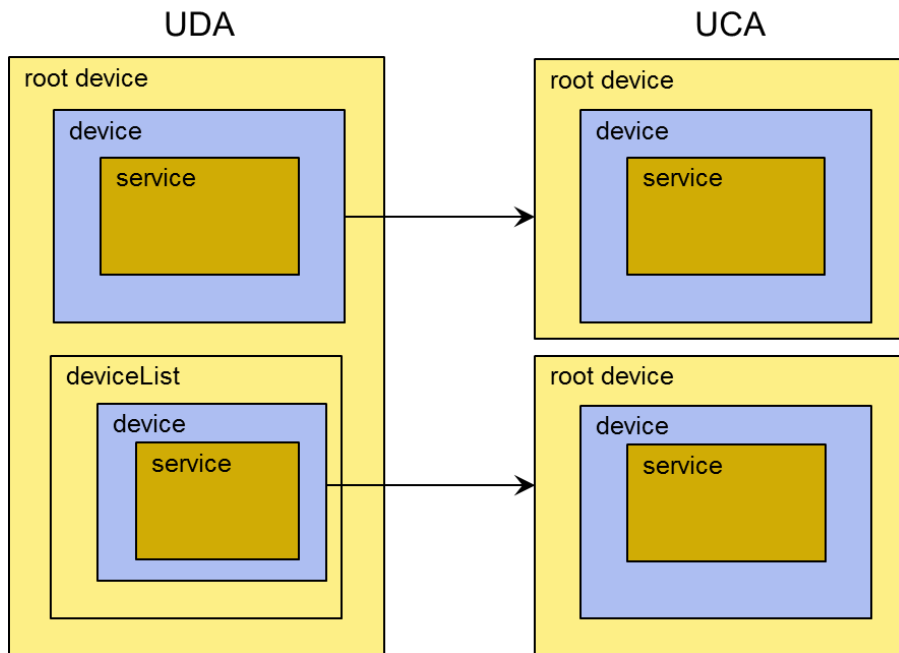


## C.5.5 Embedded Devices

UCA does not support embedded devices as described in 2. All UCEDs that advertise embedded devices in their UDA DDD shall, for each embedded device (sub-element `<deviceType>` of DDD element `<deviceList>`), connect and bind a separate **full JID** on its cloud interface using that `<deviceType>` and the `UDN` of the related embedded device (see

Figure C-7). Each embedded device advertised on the UCA interface shall have the <root> element and <specVersion> element added so that it is a conforming DDD.

Figure C-7: — UDA to UCA Mapping of embedded devices



## C.6 Presence and Discovery

### C.6.1 Presence (Analog to NOTIFY with sdp:alive)

A UCCD shall announce its availability by sending an XMPP `<presence>` stanza with no `@type` attribute which is interpreted as "available", and with the `<uc>` element (see [RFC-6121] section 4.7.3 Extended Content) advertizing the current UCCD configuration. This is the analog of a device NOTIFY with `sdp:alive`.

The template for the UCCD `<presence>` stanza is:

```
<presence>
  <uc xmlns="urn:schemas-upnp-org:cloud-1-0">
    <configIdCloud hash="uda sha-256">vendor calculated value
      as described below
    </configIdCloud>
  </uc>
</presence>
```

`<presence>`

Required. Shall be implemented according to [RFC-6120]. Case sensitive.

`<uc>`

Required. Type is `<XML>`. Single valued. Shall have "`urn:schemas-upnp-org:cloud-1-0`" as the value for the `xmlns` attribute; this references the UPnP Cloud Schema (described below). Case sensitive. Shall have one of the following values.

`<configIdCloud>`

Required. Type is `<XML>`. Single Valued. Case sensitive. Value shall be `xsd:string` with either a value equivalent to the `configId` value (see 2.3) or a Base64 encoding of the SHA-256 hash (see usage in [DP]) calculated over the UCCD advertised DDD concatenated with all the UCCD advertised

SCPDs (if present) in the order they appear in the UCA exposed DDD, concatenated with a Base64 encoding of each icon binary in the order they appear in the UCA exposed DDD [iconList](#) element (if present). That is SHA-256(<DDD xml><SCPD-1 xml> . . . <SCPD-N xml>Base64-icon1binary . . . Base64-iconNbinary).<sup>16</sup>

Note, the hash calculation is for self-consistency between device versions and is not intended for verifying software versions. However, no additional characters shall be included between the concatenated DDDs, SCPDs, and Base64 encoded icon binaries, that is, the hash is calculated over a single character string.

@hash

Required. xsd:string. Case sensitive. The following values are defined:

"[uda](#)" indicates that the [configIdCloud](#) value is the xsd:string equivalent to the [configId](#) xsd:int value (see 2.3).

"[sha-256](#)" indicates that the calculated hash of the device description is calculated using the SHA-256 algorithm [Ref].

Note that this template, or any other template in the UCA, does not prohibit the use of any additional, valid XML components, especially, those defined by the XMPP Standards Foundation (XSF), as long as they do not directly contradict the values defined in the UCA and are properly identified by valid XML namespaces. However, do not expect UCCDs and UCC-CPs to understand XMPP messages embedded in UCA specific elements.

Continuing the example from above the [MediaServer:4](#) UCCD would send a [presence](#) stanza as follows:

```
UCCD:jeffrey@mycloud.org/urn:schemas-upnp-  
org:device:MediaServer:4:uuid:e70e9d0e-d9eb-4748-b163-636a323e7950
```

```
<presence>  
  <uc  
    xmlns="urn:schemas-upnp-org:cloud-1-0">  
      <configIdCloud hash="sha-256">  
        jNwTQLAos+iQRmkykrNHk6YDvqxcCSP6dF8FZ1VhXBA=  
      </configIdCloud>  
    </uc>  
</presence>
```

A UCCD announcing initial [presence](#) will also need to configure their icon [metadata/data](#) and event [PubSub](#) as described later in this Annex.

A UCC-CP shall announce its [presence](#) (essentially for discovering UCCDs) by sending a [presence](#) stanza with no [type](#) attribute, which is interpreted as "[available](#)", as described in the template below:

```
<presence/>
```

```
<presence>
```

Required. Shall be implemented according to [RFC-6120]. Case sensitive.

A UCS shall broadcast the received UCCD or UCC-CP [presence](#) stanza to all *subscribed* UCCDs and UCC-CPs connected to the users account and add the [to](#) and [from](#) attributes as indicated below.

---

<sup>16</sup> Examples show use of hash="[sha-256](#)".

From [RFC-6121] section 4.2.2

The user's server (US) MUST also broadcast initial presence from the user's newly "available" resource to all of the user's "available" resources, including the resource that generated the presence notification in the first place (i.e., an entity is implicitly subscribed to its own presence).

The template for the outgoing UCS presence stanza for a UCCD is:

```
<presence
  from="localpart@domainpart/resourcepart of UCCD sending presence
        conforming to section C.5.4"
  to="localpart@domainpart/resourcepart of UCCD or UCC-CP subscribed
        to presense conforming to section C.5.4">
  <uc xmlns="urn:schemas-upnp-org:cloud-1-0">
    <configIdCloud hash="sha-256">vendor calculated value as described below
    </configIdCloud>
  </uc>
</presence>
```

and for a UCC-CP is:

```
<presence
  from="localpart@domainpart/resourcepart of UCC-CP sending presence
        conforming to section C.5.4"
  to="localpart@domainpart/resourcepart of UCCD or UCC-CP subscribed
        to presense conforming to section C.5.4">
</presence>
```

Note that the @from and @to attributes are not included in the UCC-CP and UCCD presence stanzas to the UCS but are added by the UCS to the outgoing presence stanzas as follows:

- @from  
Required. Shall be implemented according to [RFC-6120], that is, only when multiple resources are implemented. Shall have a value of the full JID of the UCCD or UCC-CP announcing presence. Case sensitive (for resource part only).
- @to  
Recommended. Shall be implemented according to [RFC-6120]. Shall have a value of the JID of the subscribed UCCD or UCC-CP. Case sensitive (for resource part only).

Continuing the previous example, and assuming for now the MediaServer:4 UCCD is the only UCCD or UCC-CP currently defined for the jeffrey account, the server would send the single presence stanza below:

```
UCS:mycloud.org
<presence
  to="jeffrey@mycloud.org/urn:schemas-upnp-org:device:
        MediaServer:4:uuid:e70e9d0e-d9eb-4748-b163-636a323e7950"
  from="jeffrey@mycloud.org/urn:schemas-upnp-org:device:
        MediaServer:4:uuid:e70e9d0e-d9eb-4748-b163-636a323e7950">
  <uc xmlns="urn:schemas-upnp-org:cloud-1-0">
    <configIdCloud hash="sha-256">
      jNwTQLAos+iQRmkykrNHk6YDvqxcCSP6dF8FZ1VhXBA=
    </configIdCloud>
  </uc>
</presence>
```

A second UCCD is now added to jeffrey@mycloud.org account. A MediaRender:3 UCCD with an embedded, second MediaRenderer:3 device. On the local network side it is advertised with a configId value of 3. On the cloud interface it will be advertised as two

separate devices. Assuming the two devices have already executed a proper **resource** bind as described in section C.5.4, the following presence exchange will occur.

*MediaRenderer:3* instance 1 will send its presence. The **presence** will be broadcast to all "available" **resources** on `jeffrey's mycloud.org` account. In this case, to UCCD *MediaServer:4* and UCCD *MediaRenderer:3* instance one.

Note that the **configIdCloud** element value will be calculated on the DDD, SCPDs, and icon binaries of the individual advertised UCCD instance and will be different for the two resulting *MediaRenderer:3* devices (as shown in the two **presence** examples below).

```
UCCD:jeffrey@mycloud.org/urn:schemas-upnp-  
org:device:MediaRenderer:3:uuid:0ee79d0e-e8f5-80ca-4123-225886a58850  
(UDA root device or instance 1)
```

```
<presence  
  <uc xmlns="urn:schemas-upnp-org:cloud-1-0">  
    <configIdCloud hash="sha-256">  
      VgjhrqT2VWH0OXHap/rZNkiJc/hQbztgL/EsSaGttng=  
    </configIdCloud>  
  </uc>  
</presence>
```

```
UCCD:jeffrey@mycloud.org/urn:schemas-upnp-  
org:device:MediaRenderer:3:uuid:88509d0e-e8f5-80ca-4123-225886a50ee7  
(UDA embedded device local side or instance 2)
```

```
<presence>  
  <uc xmlns="urn:schemas-upnp-org:cloud-1-0">  
    <configIdCloud hash="sha-256">  
      NXVQEoeOcbQt4NKwPyMDT26ml0VMAjP8IEgM5aHu7iA=  
    </configIdCloud>  
  </uc>  
</presence>
```

A UCC-CP is now connected and its **presence** is sent to the UCS as:

```
UCC-CP:jeffrey@mycloud.org/urn:schemas-upnp-org:cloud-1-  
0:ControlPoint:1:ad93e8f5-634b-4123-80ca-225886a5c0e8  
<presence/>
```

and subsequently sent to each subscribed UCCD and UCC-CP with the appropriate **@to** and **@from** attributes added.

The cloud discovery (analog to NOTIFY) behavior for UCCDs and UCC-CPs can be summarized as:

- UCCDs and UCC-CPs bind a **resource** as described in section C.5.4 of this specification.
- UCCDs, after successfully binding a conforming **resource**, announce their **presence** with a **<presence>** stanza with no **@type** attribute, which is interpreted as "available", and whose body conforms to section C.6.1 of this specification applying to UCCDs.
- UCC-CPs, after successfully binding a conforming **resource**, announce their **presence** with a **<presence>** stanza with no **@type** attribute, which is interpreted as "available", and whose body conforms to section C.6.1 of this specification applying to UCC-CPs.

The cloud discovery behavior (analog to multicast of the NOTIFY) of an XMPP cloud server supporting the UPnP protocol capability can be summarized as:

- A UCS broadcast any **<presence>** stanza with no **@type** attribute, which is interpreted as "available", received from any UCCD or UCC-CP **resource** to all connected and

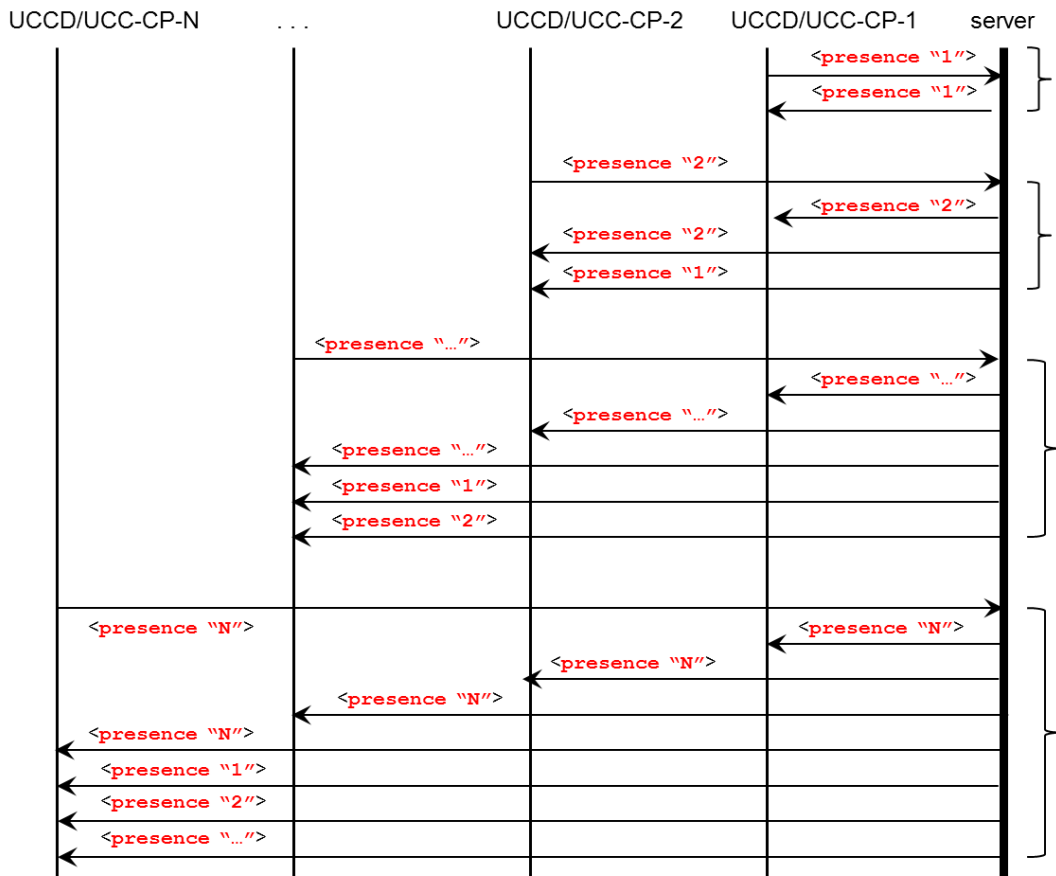
"available" UPnP UCCD and UCC-CP resources with the <presence> stanza body unaltered.

- A UCS broadcast the last presence with @type attribute value of "unavailable" received from any UCCD or UCC-CP resource to all connected and "available" UPnP UCCDs and UCC-CPs with the <presence> stanza body intact.

The individual presence exchange between the UCCDs, UCC-CPs, and UCS for an N connected UPnP scenario is illustrated in

Figure C-8.

Figure C-8: — Self <presence> stanza flows



### C.6.2 XMPP disco#items (analog to M-SEARCH for users UCCDs and UCC-CPs)

A UCC-CP shall support sending an <iq> query with @type attribute value of "get" and a <query> element with @xmlns attribute value of "http://jabber.org/protocol/disco#items" to its UCS as defined in [XEP-0030]. A UCC-CP may send this query at any time that it needs to discovery "available" UCCDs. A UCC-CP shall be able to distinguish UCA conforming resources from other resources.

The example session is continued below.

```
UCC-CP:jeffrey@mycloud.org/urn:schemas-upnp-org:cloud-1-0:ControlPoint:1:ad93e8f5-634b-4123-80ca-225886a5c0e8
```

```
<iq
  to="jeffrey@mycloud.org"
  id="get uccds and ucccps 1"
  type="get">
  <query xmlns="http://jabber.org/protocol/disco#items"/>
</iq>
```

The UCS responds with the four "available" UCCDs and UCC-CPs already connected. Note that this is only useful for discovering connected UPnP entities for a specific user.

```
UCS:mycloud.org
```

```
<iq
  to="jeffrey@mycloud.org/urn:schemas-upnp-org:cloud-1-0:ControlPoint:1:ad93e8f5-634b-4123-80ca-225886a5c0e8"
  from="jeffrey@mycloud.org"
  id="get uccds and ucccps 1"
  type="result">
  <query xmlns="http://jabber.org/protocol/disco#items"/>
    <item jid="jeffrey@upnp.org/urn:schemas-upnp-org:cloud-1-0:ControlPoint:1:ad93e8f5-634b-4123-80ca-225886a5c0e8"/>
    <item jid="jeffrey@upnp.org/urn:schemas-upnp-org:device:MediaRenderer:3:88509d0e-e8f5-80ca-4123-225886a50ee7"/>
    <item jid="jeffrey@upnp.org/urn:schemas-upnp-org:device:MediaRenderer:3:0ee79d0e-e8f5-80ca-4123-225886a58850"/>
    <item jid="jeffrey@upnp.org/urn:schemas-upnp-org:device:MediaServer:4e70e9d0e-d9eb-4748-b163-636a323e7950"/>
  </query>
</iq>
```

A UCS shall support the XMPP Service Discovery protocol as specified in [XEP-0030].

A UCS shall include a `<feature var="http://upnp.org/protocol/cloud/v/">` in their response to an XMPP `disco#info` query verifying the UCA version supported by the UCS. For this specification the value of `v` shall be "1.0".

### C.6.3 Presence update (analog to NOTIFY with sssdp:update)

If a UCCD is modified in any manner that results in a change in its DDD or any of its SCPDs or any of its icon binaries (for devices that implement UDA 1.1 or higher this is equivalent to a change in the `configId`), then it shall issue a new `<presence>` stanza with the new `configIdCloud` element value. This is analog to an NOTIFY with "ssdp:update". The `<presence>` stanza shall also conform to the template in section C.6.1. A UCCD that changes only `BOOTID`(s) on the local interface shall not send an updated presence.

A UCCD updating its `presence` will also need to update its icon `metadata/data` and event `PubSub` as described later in this Annex.

For example purposes, assume the UCCD `MediaServer:4` from the previous example changes its configuration. It would send a new `<presence>` stanza much like the following:

```
UCCD:jeffrey@mycloud.org/urn:schemas-upnp-org:device:MediaServer:4:uuid:e70e9d0e-d9eb-4748-b163-636a323e7950
<presence
  <uc
    xmlns="urn:schemas-upnp-org:cloud-1-0">
    <configIdCloud hash="sha-256">
      sddlg2s+lp0E9ryWquNklHlnEuBvRQpyNc8sd0rbPpU=
    </configIdCloud>
    </uc>
</presence>
```



A UCS shall add the @[to](#) and @[from](#) attributes as previously indicated.

### C.6.4 Presence "unavailable" (Analog to NOTIFY with [ssdp:byebye](#))

At any point, a UCCD or UCC-CP may make itself "unavailable" on its cloud interface(s) by sending a [<presence>](#) stanza with @[type](#) attribute value of "[unavailable](#)" to its UCS as described in the template below:

```
<presence
  type="unavailable" />
```

<presence>

Required. Shall be implemented according to [RFC-6120]. Case sensitive.

@type

Required. Shall be implemented according to [RFC-6120]. Shall have a value of "[unavailable](#)".

This is the analog of a NOTIFY with [ssdp:byebye](#).

Note that a UCCD or UCC-CP is not required to make itself "unavailable" on its cloud interface(s) just because it has left one or all of its locally connected interfaces, that is, it is assumed that the UCCD or UCC-CP will often keep itself alive on its cloud interfaces.

```
UCCD:jeffrey@mycloud.org/urn:schemas-upnp-
org:device:MediaServer:4:uuid:e70e9d0e-d9eb-4748-b163-636a323e7950
```

```
<presence
  type="unavailable" />
```

A UCS shall add the @[to](#) and @[from](#) attributes as previously indicated.

### C.6.5 Service Level Discovery

Currently UCA does not include a mechanism to provide filtered Discovery, for example, finding a particular service among "available" UCCDs. This is primarily because a significant amount of information is implied by the [full JID](#) of the UCCD and the trade-off of having additional granularity would not compensate for the additional UCS traffic. Instead, it is highly recommended that a UCC-CP cache the [configIdCloud](#) value of each UCCD along with the devices DDD, SCPDs, and icons (if advertised). If, upon subsequent received [presence](#), the [configIdCloud](#) value has changed the UCC-CP should check to see if services have been updated and re-cache as needed. Also, if vendor defined UCCDs are encountered; UCC-CPs should attempt to learn the services provided by these UCCDs and associate them with future [presence](#) of similar UCCDs.



Chattiness – pings! It is recommended to use XMPP ping sparingly; a ping rate of 30 minutes is recommended.

### C.6.6 IQ:Query for DDD and SCPD Exchange (analog of HTTP GET for DDD and SCPD)

#### C.6.6.1 DDD and SCPD "get"

A UCC-CP shall retrieve UCCD DDDs and SCPDS using the [<iq>](#) stanza with the @[type](#) attribute value of "[get](#)" as described in the following template. This is the analog of using HTTP GET to exchange DDDs and SCPDs in the UDA.

```
<iq
  id="vendor defined value"
```

```

to="localpart@domainpart/resourcepart of UCCD conforming to section C.5.4"
from="localpart@domainpart/resourcepart of UCC-CP conforming to section
C.5.4"
type="get">
<query xmlns="urn:schemas-upnp-org:cloud-1-0"
type="description"
name="value of UDN"/>
</iq>

```

<iq>

Required. Shall be implemented according to [RFC-6120]. Case sensitive.

@id

Required. Shall be implemented according to [RFC-6120]. Case sensitive.

@from

Allowed. If present, shall be implemented according to [RFC-6120]. Shall have a value of the **full JID** of the UCC-CP requesting a UCCDs' DDD and SCPDs description. Case sensitive. Note, @from will be added by UCS to outgoing stanza.

@to

Required. Shall be implemented according to [RFC-6120]. Shall have a value of the **full JID** of the UCCD for which a UCC-CP is requesting a DDD and SCPDs description. Case sensitive.

@type

Required. Type is string. Shall have a value of **get**.

<query>

Required. Type is <XML>. Shall have "urn:schemas-upnp-org:cloud-1-0" as the value for the xmlns attribute; this references the UPnP Cloud Schema (described in section C.11). Case sensitive.

@type

Required. Type is xsd:string. Shall have a value of **description**.

@name

Required. Type is xsd:string. Shall have a value conditioned on the @type attribute value where the value of **name** shall be the value of the **UDN** of the target (**to**) UCCD.

Note that since only one UDA device is "available" in a UCCD there is no need to include any hierarchy in the DDD/SCPD **iq** request stanza.

### C.6.6.2 DDD and SCPD "result" or "error"

A UCCD receiving a valid UCC-CP request as described in section C.6.6.1 shall return an **iq** stanza with the @type attribute value of **result**, assuming no other XMPP related errors are detected, and include the **query** child element with either an @type attribute value of **described** or **error** and the associated device XML or UCA errorCode as described in the template below. This is the analog of the HTTP GET response of UDA with the DDD XML and SCPD XML in the response body.

```

<iq
id="vendor defined value"
from="localpart@domainpart/resourcepart of UCCD conforming
to section C.5.4"
to="localpart@domainpart/resourcepart of UCC-CP conforming
to section C.5.4"
type="result"
<query xmlns="urn:schemas-upnp-org:cloud-1-0"
type="described | error"
name="received value of UDN"/>
Concatenated DDD XML, SCPDs
or
UPnP Cloud error Description

```

```
</query>  
</iq>
```

<iq>

Required. Shall be implemented according to [RFC-6120]. Case sensitive.

@id

Required. Shall be implemented according to [RFC-6120]. Case sensitive.

@from

Allowed. If present, shall be implemented according to [RFC-6120]. Shall have a value of the [full JID](#) of the UCCD for which a UCC-CP is requesting a DDD and SCPDs description. Case sensitive. Note, @from will be added by UCS to outgoing stanza.

@to

Required. Shall be implemented according to [RFC-6120]. Shall have a value of the [full JID](#) of the UCC-CP requesting a UCCDs' DDD and SCPDs description. Case sensitive.

@type

Required. Shall be implemented according to [RFC-6120]. Shall have a value of "[result](#)" assuming no XMPP or UCA errors, otherwise shall have a value of "[error](#)" and resulting body according to [RFC-6120] or UCA error as described below.

<query>

Required. Type is <XML>. Shall have "urn:[schemas-upnp-org:cloud-1-0](#)" as the value for the xmlns attribute; this references the UPnP Cloud Schema (described below). Case sensitive.

@type

Required. Type is xsd:string. Shall have a value of "[described](#)" or "[error](#)" according to description for value of <query> element below.

@name

Required. Type is xsd:string. Shall have a value identical to @[name](#) attribute value in the <[iq](#)> request.

value of <query> element

Required. Type is XML. Value is dependent on the <[iq](#)> request as described below.

If the received [query@type](#) attribute is [description](#) and the received [query@name](#) attribute value matches with the UCCD [UDN](#) element, then the value of the [query](#) element shall be the device  
DDD                    XML                    (as                    illustrated                    in

Figure C-7) followed by all SCPD XML for all services in the order they appear in the device DDD XML. The response shall not include the `<?xml version="1.0"?>` element. The response shall not include a `deviceList` element. Note that in this case, the returned `query@type` attribute value shall be `described`.

If the received `query@type` attribute is `description` and the received `query@name` attribute value does not match with the UCCD `UDN` element, then the value of the `query` element shall be


```
<UPnPError xmlns="urn:schemas-upnp-org:control-1-0">
  <errorCode>900</errorCode>
  <errorDescription>deviceType mismatch</errorDescription>
</UPnPError>
```

Note that in this case, the returned `query@type` attribute value shall be `error`. See section C.10 for further description.

Note that the XML shall be escaped as described in the UDA.

UCS shall support TLS compression, as well as, basic compression of XMPP stanzas using `zlib` as defined in [XEP-0138].

UCCDs and UCC-CPs are highly recommended to support TLS compression, as well as, basic compression of XMPP stanzas using `zlib` as defined in [XEP-0138]. When BOSH [XEP-0138] is used, TLS and stanza compression should not be used and instead HTTP level compression negotiated.

	<p>Typically DDD and SCPD XML using the recommended compression will achieve compression efficiencies in the 75 to 90 percent range for XMPP stanzas. It is highly recommended that the UCCDs, UCC-CPs and UCSs for this version, "1.0", support a minimum of 64 Kbyte stanzas.</p>
---	---

The control point from the continuing example requests the *MediaServer:4* DDD.

```
UCC-CP:jeffrey@mycloud.org/urn:schemas-upnp-org:cloud-1-0:ControlPoint:1:ad93e8f5-634b-4123-80ca-225886a5c0e8
```

```
<iq id="get-MediaServer-ddd" to="jeffrey@mycloud.org/urn:schemas-upnp-org:device:MediaServer:4:uuid:e70e9d0e-d9eb-4748-b163-636a323e7950" type="get">
  <query xmlns="urn:schemas-upnp-org:cloud-1-0" type="description" name="uuid:e70e9d0e-d9eb-4748-b163-636a323e7950"/>
</iq>
```

Note that the `<iq>` stanza is relayed through the UCS.

```
UCCD:jeffrey@mycloud.org/urn:schemas-upnp-org:device:MediaServer:4:uuid:e70e9d0e-d9eb-4748-b163-636a323e7950
```

```
<iq id="get-MediaServer-ddd" to="jeffrey@mycloud.org/urn:schemas-upnp-org:device:MediaServer:4:uuid:e70e9d0e-d9eb-4748-b163-636a323e7950" type="result">
  <query xmlns="urn:schemas-upnp-org:cloud-1-0" type="described" name="uuid:e70e9d0e-d9eb-4748-b163-636a323e7950"/>
</iq>
```

```

<root xmlns="urn:schemas-upnp-org:device" configId="2">
  <specVersion>
    <major>1</major>
    . . .
  </specVersion>
  <device>
    <deviceType>urn:schemas-upnp-org:device:
      MediaServer:4</deviceType>
    . . .
    <iconList>
      . . .
    </iconList>
    <serviceList>
      . . .
    </serviceList>
  </device>
</root>
<scpd
  xmlns="urn:schemas-upnp-org:service-1-0"
  . . .
  configId="2">
  <specVersion>
    <major>4</major>
    . . .
  </specVersion>
  <actionList>
    <action>
      . . .
    </action>
  </actionList>
</scpd>
. . .
<scpd
  xmlns="urn:schemas-upnp-org:service-1-0"
  . . .
  configId="2">
  <specVersion>
    <major>3</major>
    . . .
  </specVersion>
  <actionList>
    <action>
      . . .
    </action>
  </actionList>
</scpd>
</query>
</iq>

```

### C.6.6.3 Exchange of Device Icons

A UCS shall support [XEP-0084].

A UCCD that includes the [iconList](#) element in its DDD shall support [XEP-0084] and map the conforming [XEP-0084] [data](#) and [metadata](#) elements as defined in Table C-2.

**Table C-2: — Mapping of DDD [iconList](#) to [XEP-0084]**

DDD element	XEP-0084	Explanation
< <a href="#">iconList</a> >	<a href="#">metadata</a>	The <a href="#">icon</a> elements in the <a href="#">iconList</a> and <a href="#">info</a> elements in the <a href="#">metadata</a> element have a 1:1

		mapping.
<u>&lt;icon&gt;</u>	<u>info</u>	The first <u>icon</u> sub-element in the UCCD <u>iconList</u> element shall correspond to the "image/png" icon whose <u>&lt;data&gt;</u> element is uploaded to the UCS. That is, the first <u>info</u> element shall have a MIME type of "image/png" and its Base64 encoded binary shall fit within the 64K stanza size limit. The Base64 binary, shall be uploaded to the UCS as the <u>data</u> element (see below in this table).
	<u>info@id</u>	see [XEP-0084]
<u>&lt;mimetype&gt;</u>	<u>info@type</u>	Required. Shall indicate the MIME type of the first icon. Shall be indicative of MIME type "image/png".
<u>&lt;width&gt;</u>	<u>info@width</u>	Required. Shall be the same, equivalent integer value as the height of the first icon.
<u>&lt;height&gt;</u>	<u>info@height</u>	Required. Shall be the same, equivalent integer value as the width of the first icon.
<u>&lt;depth&gt;</u>	NA	Allowed in the UCA <u>iconList</u> element. Not allowed as a <u>metadata</u> sub-element.
<u>&lt;url&gt;</u>	<u>info@url</u>	Allowed. Shall be the URL provided by the avatar metadata as described in [XEP-0084] for this icon.
NA	<u>data</u>	Base64 encoded binary of the first icon.

A UCCD shall publish the corresponding icon data element as described in the following template.

```
<iq
  id="vendor defined value"
  from="localpart@domainpart/resourcepart of UCCD conforming
        to section C.5.4"
  to=" PubSubName of UCS supporting UCA
      (see footnote in section C.6.6.3)"
  type="set">
  <pubsub xmlns="http://jabber.org/protocol/pubsub"
    <publish node="urn:xmpp:avatar:data">
      <item id="vendor SHA-1 Hash as described in XEP-0084">
        <data xmlns="urn:xmpp:avatar:data">
          BASE 64 encoded binary for first icon"
        </data>
      </item>
    </publish>
  </pubsub>
</iq>
```

<iq>, iq@id, iq@from, iq@to<sup>17</sup>. Required. See equivalent in previous template for <iq> attributes<sup>18</sup>.

@type

Required. Shall be implemented according to [RFC-6120]. Shall have a value of "set".

<pubsub>

<sup>17</sup> For all "to" elements in the UCA, the "to" element is not required if the "PubSub" supports Personal Eventing Protocol as referenced in [XEP-0084]; however, UCCDs and UCC-CPs must be able to recognize this support on the UCS.

<sup>18</sup> Note that for brevity, previously defined template entries are included by reference. For example, all entries described as "PubSubName of UCS supporting UCA" have the same requirements. New elements or changed elements will be included in the template.

Required. Shall be implemented according to [XEP-0084] and contain a [data](#) element corresponding to the first [iconList icon](#) binary.

A UCCD shall publish the corresponding icon [metadata](#) element as described in the following template.

```
<iq
  id="vendor defined value"
  from="localpart@domainpart/resourcepart of UCCD conforming
    to section C.5.4"
  to=" PubSubName of UCS supporting UCA
    (see footnote in section C.6.6.3)"
  type="set">
  <pubsub xmlns="http://jabber.org/protocol/pubsub">
    <publish node="urn:xmpp:avatar:metadata">
      <item id="vendor calculated SHA-1 Hash as described
        in XEP-0084 corresponding to first icon binary">
        <metadata xmlns="urn:xmpp:avatar:metadata">
          <info
            id="vendor calculated SHA-1 Hash value as described
              in XEP-0084"
            type="vendor calculated image MIME type value
              as described in XEP-0084"
            width="vendor calculated image width value
              as described in XEP-0084"
            height="vendor calculated image height value
              as described in XEP-0084"
            bytes="vendor calculated image size in bytes value
              as described in XEP-0084"
            url="vendor supplied url for additional icon
              binary retrieval
              as described in XEP-0084"/>
          </metadata>
        </item>
      </publish>
    </pubsub>
  </iq>
```

<iq>, iq@id, iq@from, iq@to. Required. See equivalent in previous template for <iq> attributes.

@type

Required. Shall be implemented according to [RFC-6120]. Shall have a value of "[set](#)".

<pubsub>

Required. Shall be implemented according to [XEP-0084] and contain a [metadata](#) element and at least one [info](#) element corresponding to the first [iconList icon](#) element.

A UCCD publishes its [iconList icon](#) XEP-0084 [data](#) and [metadata](#) to the UCS as shown in the following example.

UCCD:jeffrey@mycloud.org/urn:schemas-upnp-  
org:device:MediaServer:4:uuid:e70e9d0e-d9eb-4748-b163-636a323e7950

```
<iq
  id="pub-main-icon-data"
  to="pubsub.mycloud.org"
  type="set">
  <pubsub>
    <publish node="urn:xmpp:avatar:data">
      <item id="299bc6f8e9ef9066971f111f4b3c50d7b0df729d">
        <data xmlns="urn:xmpp:avatar:data">
          aOIHdsbjfsojjsfHOIHafoj...
        </data>
      </item>
    </publish>
  </pubsub>
</iq>
```

```
        </item>
      </publish>
    </pubsub>
  </iq>

P-USC:pubsub.mycloud.org (PubSub service affiliated with UCS
mycloud.org)
<iq
  id="pub-main-icon-data"
  to="jeffrey@mycloud.org/urn:schemas-upnp-org:device:
    MediaServer:4:uuid:e70e9d0e-d9eb-4748-b163-636a323e7950"
  type="result">
</iq>

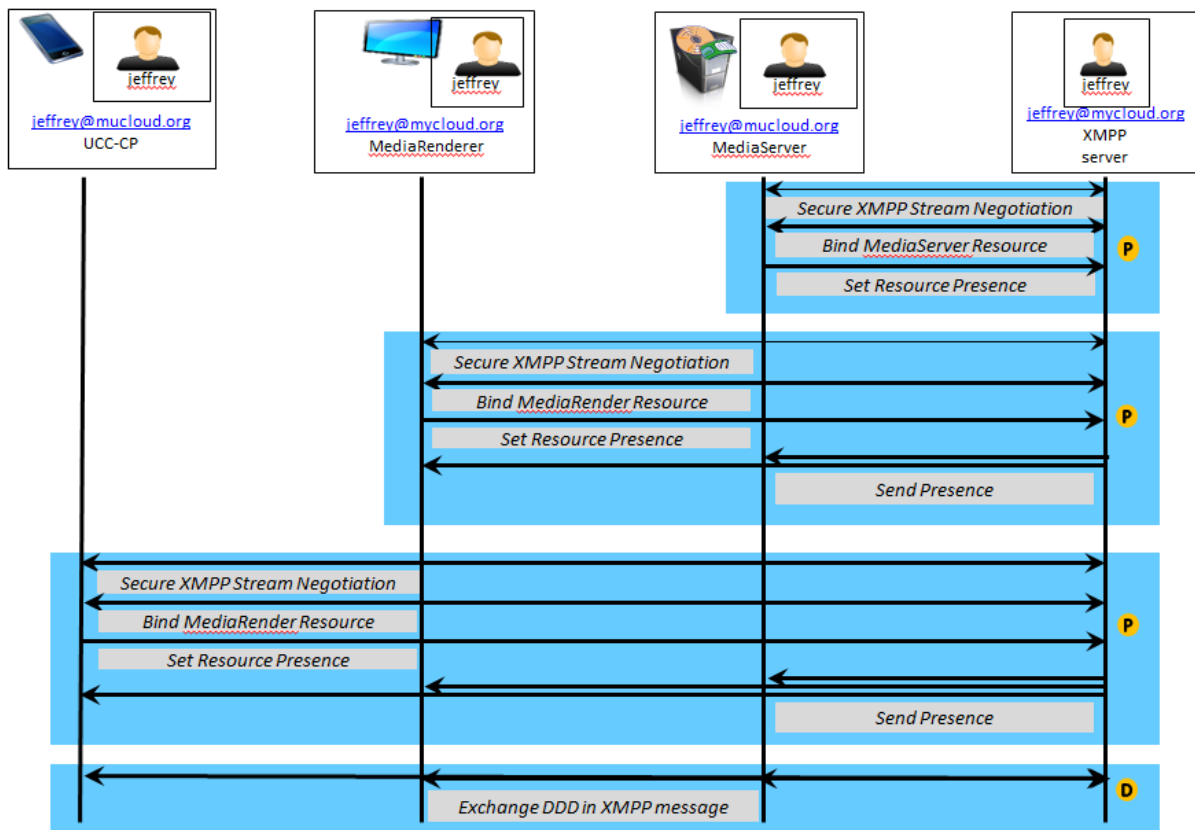
UCCD:jeffrey@mycloud.org/urn:schemas-upnp-
org:device:MediaServer:4:uuid:e70e9d0e-d9eb-4748-b163-636a323e7950
<iq
  id="pub-main-icon-metadata"
  to="pubsub.mycloud.org
  type="set">
  <pubsub>
    <publish node="urn:xmpp:avatar:metadata">
      <item id="299bc6f8e9ef9066971f111f4b3c50d7b0df729d">
        <metadata xmlns="urn:xmpp:avatar:metadata">
          <info id="299bc6f8e9ef9066971f111f4b3c50d7b0df729d"
            type="image/png"
            bytes="12345"
            type="image/png"
            height="48"
            width="48"/>
          </metadata>
        </item>
      </publish>
    </pubsub>
  </iq>

P-USC:pubsub.mycloud.org (PubSub service affiliated with UCS
mycloud.org)
<iq
  id="pub-main-icon-data"
  to="jeffrey@mycloud.org/urn:schemas-upnp-org:device:
    MediaServer:4:uuid:e70e9d0e-d9eb-4748-b163-636a323e7950"
  type="result"/>
```

Figure C-9 represents the above described operations in a multi-device scenario.

**Figure C-9: — Combined Connect, Announce and Describe Message Flow**





**Table C-3: — Summary of Requirements for DDD elements**

DDD element	R/A/CR/CA/P <sup>19</sup>	Explanation
<?xml>		
<root>	R	see 2.3
root@configId	R	
<specVersion>	R	
<major>	R	
<minor>	R	
<URLBase>	P	
<device>	R	
<deviceType>	R	
<friendlyName>	R	
<manufacturer>	R	
<manufacturerURL>	A <sup>20</sup>	If included, shall not create a security risk, Should be a publicly available URL.
<modelDescription>	A	

<sup>19</sup> R = Required, A = Allowed, CR = Conditionally Required (is a required sub-element of an allowed element), Conditionally Allowed (is an allowed sub-element of a an optional element), NA = Not Applicable. P = Prohibited (is allowed in the UDA but not in the UCA or has been deprecated in the UDA).

<sup>20</sup> The following security considerations should be considered when exposing the manufacturerURL and modelURL. A URL exposed directly from the UCCD should be secure as described for presentation URL, otherwise, it shall be non-local to the device; for example http://vendor.com/device-model-a.

<modelName>	R	
<modelURL>	A	
<serialNumber>	A	
<UDN>	R	
<UPC>	A	
<iconList>	R	see C.6.6.3 along with 2.3
<icon>	A	
<mimetype>	CR	
<width>	CR	
<height>	CR	
<depth>	CR	
<url>	NA	Shall be the URL provided by the avatar metadata as described in [XEP-0084] for this icon
<serviceList>	A	The order of the attached SCPDs shall be the same as the order of the <service> elements in the <serviceList>
<service>	CA	
	CR	
<serviceType>		
<serviceId>	CR	
<SCPDURL>	NA	Not used in UCA, shall be ignored
<controlURL>	NA	Not used in UCA, shall be ignored
<eventURL>	NA	Not used in UCA, shall be ignored
<deviceList>	P	Embedded devices are not allowed in UCA
<device>	P	Embedded devices are not allowed in UCA
<presentationURL>	A	Allowed. If present, shall not create a security risk; for example, it can be an HTTPS URL. May be publicly available or reachable via other XMPP protocols, such as [XEP-0332]. Shall contain an HTML document as described in the 5. Shall be an absolute URL (note that this overrides the relative URL as described in UDA which shall be ignored).

## C.7 PubSub (Analog of Eventing)

UCA uses the XMPP Publication-Subscribe (PubSub) service [XEP-0060] as the cloud analog to UPnP eventing. A PubSub service is similar to event subscribe (section 4.1.2) and unsubscribe (section 4.1.4), however an intermediate entity (the PubSub service running on a PubSub server, which could be integrated with the UCS) is used to decouple the event communication component (publish) from the subscription management and event

communication component (subscribe). In general, a [PubSub](#) service provides greater scalability and a more flexible configuration. A brief overview extracted from [XEP-0060] is provided next describing some basic concepts of the XMPP [PubSub](#) service, namely nodes and the related access models. Later, additional requirements related to supporting UCA [PubSub](#) are defined.

XMPP defines two types of publication [node](#), Leaf and Collection. These are described in Table C-4.

**Table C-4: — [PubSub Node](#) Types**

Node Type	Description
Leaf	A node that contains published items only. It is NOT a container for other nodes. This is the most common node type.
Collection	A node that contains nodes and/or other collections but no published items. Collections make it possible to represent more sophisticated relationships among nodes. For details, refer to [XEP-0248].

XMPP defined nodes can have access models as described in Table C-5. They are described in order of most open to least open.

**Table C-5: — [PubSub Node](#) Access Models**

Access Model	Description
Open	Any entity may subscribe to the node (i.e., without the necessity for subscription approval) and any entity may retrieve items from the node (i.e., without being subscribed); this SHOULD be the default access model for generic pubsub services.
Presence	Any entity with a subscription of type "from" or "both" may subscribe to the node and retrieve items from the node; this access model applies mainly to instant messaging systems (see [RFC-6121]).
Roster	Any entity in the specified roster group(s) may subscribe to the node and retrieve items from the node; this access model applies mainly to instant messaging systems (see [RFC-6121]).
Authorize	The node owner must approve all subscription requests, and only subscribers may retrieve items from the node.
Whitelist	An entity may subscribe or retrieve items only if on a whitelist managed by the node owner. The node owner MUST automatically be on the whitelist. In order to add entities to the whitelist, the node owner SHOULD use the protocol specified in the "Manage Affiliated Entities" section of [XEP-0060], specifically by setting the affiliation to "member".

The access model is assigned by the node [owner](#) during its configuration. Support for the "owner" and "none" affiliations are required by [XEP-0060]. Support for all other affiliations (as profiled in Table C-6) is recommended. For each non-required affiliation supported by an implementation, it should return a service discovery feature of "name-affiliation" where "name" is the name of the affiliation, such as "member", "outcast", or "published. Particular kinds of [PubSub](#) services may enforce additional requirements (e.g., requiring support for a given non-required affiliation or for all affiliations). UCA defines some additional general principles for PubSub affiliations as described in C.7.6.

**Table C-6: — PubSub Affiliations and their Privileges to "publishing" as defined by [XEP-0060] and further restricted by UCA (see footnotes)**

Affiliation	Subscribe	Retrieve Items	Publish Items	Delete Single Item	Purge Node	Configure Node	Delete Node
Owner	Yes <sup>21</sup>	Yes	Yes	Yes	Yes <sup>22</sup>	Yes <sup>22</sup>	Yes <sup>22</sup>
Publisher	Yes	Yes	Yes	Yes <sup>23</sup>	Yes <sup>23</sup>	No	No
Publish-Only	No	No	Yes	Yes <sup>23</sup>	No <sup>23</sup>	No	No
Member	Yes	Yes	No	No	No	No	No
None	Yes	No	No	No	No	No	No
Outcast	No	No	No	No	No	No	No

The PubSub owner can modify a JIDs affiliation and thereby its access as summarized in the Table C-7.

**Table C-7: — PubSub Affiliations and their Privileges to "subscribers"**

	Outcast	None	Member	Publisher	Owner
Outcast	--	Owner removes ban	Owner adds entity to member list	Owner adds entity to publisher list	Owner adds entity to owner list
None	Owner bans entity	--	Owner adds entity to member list	Owner adds entity to publisher list	Owner adds entity to owner list
Member	Owner bans entity	Owner removes entity from member list	--	Owner adds entity to publisher list	Owner adds entity to owner list
Publisher	Owner bans entity	Owner removes entity from publisher list	n/a	--	Owner adds entity to owner list
Owner	n/a	Owner resigns	n/a	n/a	--

To identify the PubSub JID, UCC-CPs and UCCDs shall be capable of sending a disco#item and disco#info <iq> stanza to the UCS JID and returned item JIDs as described for a UCC-CP in C.6.2 and identifying the PubSub service and PubSub service features as described in [XEP-0030] and [XEP-0060]. Some additional requirements are related to disco# are also described in C.7.2. Note that a typical PubSub JID is of the form

<sup>21</sup> Any UCCD or UCC-CP belonging to a specific user account shall be considered as an "owner" for subscription and retrieval purposes to any PubSub node created by any UCCD or UCC-CP belonging to the same account.

<sup>22</sup> Only the UCCD creating a PubSub node shall be considered the "owner" of the node for publication, deletion.

<sup>23</sup> Note: A service MAY allow any publisher to delete / purge any item once it has been published to that node instead of allowing only the original publisher to remove it. This behaviour is NOT RECOMMENDED for the publish-only affiliation, which SHOULD be allowed to delete only items that the publish-only entity has published. The ways in which an entity changes its affiliation with a node are well-defined. Typically, action by an owner is required to make an affiliation state transition. Affiliation changes and their triggering actions are specified in Table C-7.

"pubsub." concatenated with the UCS JID, for example the PubSub service for "mycloud.org" would be "pubsub.mycloud.org".

For UCCDs that have evented state variables, cloud eventing proceeds along one of two possible phases depending on whether the related UCCD PubSub hierarchy needs to be created, updated, or just published to. First the UCCD needs to determine if the PubSub hierarchy exists for itself, and if it does, whether it conforms to the latest configuration.

If the hierarchy does not exist and the UCCD has evented state variables, then the UCCD creates it by:

- Creating a UCCD collection node at the UCCDs PubSub service,
- Creating a configIdCloud leaf node of the device collection node and adding to it an item of value "CONFIGURING",
- Creating service collection node(s) for each service in the UCCD and tying them to the device collection node,
- Creating evented state variable leaf node(s) for each evented state variable and tying them to their service collection node.
- Publishing, for each evented state variable, the current state as an item of the respective event leaf node.
- Changing the configIdCloud leaf node to the current version.
- Once configured, the UCCD begins publishing all new events to the PubSub collection.

Figure C-10 illustrates the related PubSub configuration flow.

If the hierarchy does exist for the evented state variables, but the UCCD confirms the configIdCloud leaf node is not up to date then it ceases any further event publication and updates the configuration by:

- changing the configIdCloud leaf node to "CONFIGURING",
- traversing each branch of the device PubSub hierarchy and purging all evented state variable item(s) of any evented state variable that the UCCD does not support in its new configuration,
- deleting any evented state variable leaf node no longer in a specific service and adding any new evented state variable(s) as a new leaf node(s),
- deleting any service collection node(s) (and first their event node(s) and event item(s) no-longer supported by the UCCD,
- creating any new service collection node(s) as described previously,
- changing the configIdCloud node to the current version.
- resuming publication of events to the PubSub.

Else

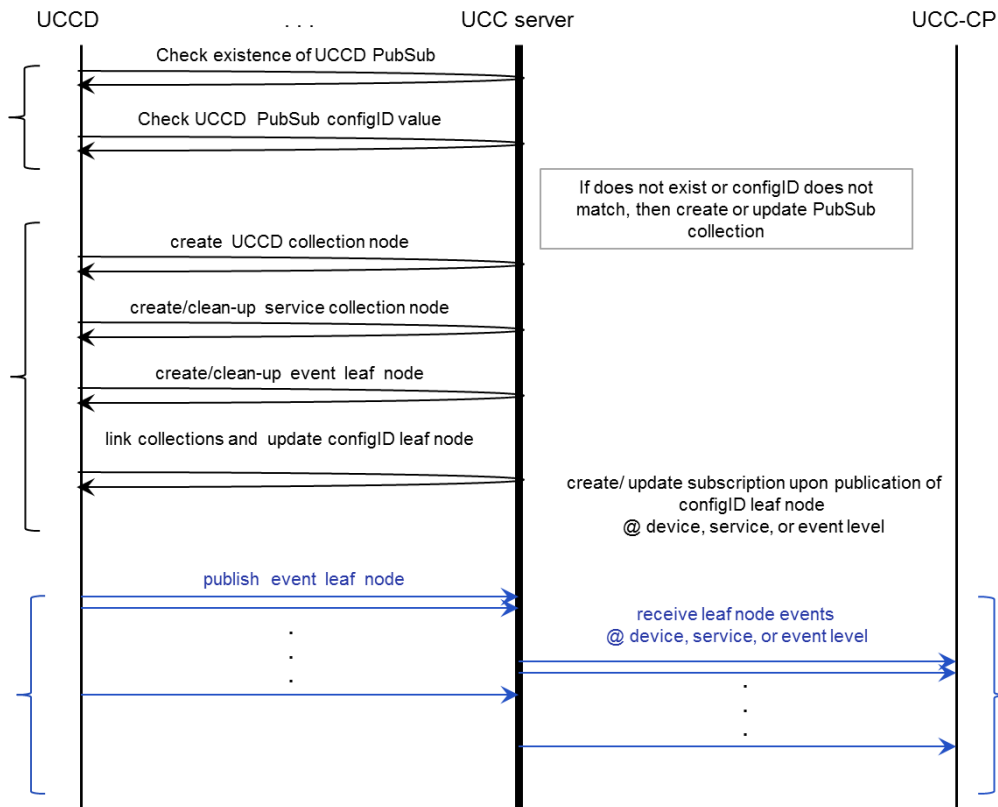
- A UCCD continues to publish events to the PubSub.

For UCC-CPs there are essentially 4 phases involved:

- A UCC-CP determines the expected PubSub structure of a UCCD from the UCCDs full JID and DDD/SCPD description.

- If the examined UCCD has evented state variables then a UCC-CP can subscribe to the PubSub service events (collection node(s) [XEP-0248] ) it needs to monitor.
- A UCC-CP monitors the PubSub service for events indicating a change (modification in the configIDCloud leaf node) to the device description and re-subscribes if necessary.
- A UCC-CP unsubscribe to events (collection node(s) or leaf node(s)) it no longer needs to monitor.

**Figure C-10: — PubSub Hierarchy Event Structure Creation**



### C.7.1 Creating the UCCD PubSub structure

A UCCD with evented state variables, before creating a PubSub collection for events, shall first identify the PubSub service availability by sending an <iq> stanza request to its UCS as described in the following template. The UCCD should verify that the PubSub supports the UPnP configuration by verifying the inclusion of the "ownerUPnP" affiliation in the access model options returned in an initial pubsub/protocol/#owner response.

```

<iq
  id="vendor defined value"
  from="localpart@domainpart/resourcepart of UCCD conforming to
        section C.5.4"
  to="PubSubName of UCS supporting UCA (see footnote in section C.6.6.3)"
  type="get"
  <query xmlns="http://jabber.org/protocol/disco#info"/>
</iq>

```

<iq>, iq@id, iq@from. Required. See equivalent in previous template for <u>iq</u> attributes.

- @to  
Required. Shall be implemented according to [RFC-6120]. Shall have a value of the name of the [PubSub](#) service supporting UCA ("PubSubName") as determined by previous [disco#items](#) and [disco#info](#) queries. Case sensitive.
- @type  
Required. Shall be implemented according to [RFC-6120]. Shall have a value of "[get](#)".
- <query>  
Required. Shall have "<http://jabber.org/protocol/disco#info>" as the value for the xmlns attribute (see schema in [XEP-0030]). Case sensitive.

A UCS shall respond to the above <iq> stanza request with the included xml <query> child element indicating support for the [PubSub](#) service conforming to [XEP-0060], that is it includes

```
<identity category="pubsub" type="service"/>
<feature var="http://jabber.org/protocol/pubsub"/>
<feature var="http://upnp.org/protocol/cloud/1.0"/>
```

The UCCD *MediaServer:4* verifies the [PubSub](#) infrastructure is supported.

UCCD:jeffrey@mycloud.org/urn:schemas-upnp-org:device:MediaServer:4:uuid:e70e9d0e-d9eb-4748-b163-636a323e7950

```
<iq
  id="check-for-pubsub"
  to="pubsub.mycloud.org"
  type="get">
  <query
    xmlns="http://jabber.org/protocol/disco#info"/>
</iq>
```

P-USC:pubsub.mycloud.org (PubSub service affiliated with UCS **mycloud.org**)

```
<iq
  id="check-for-pubsub"
  to="jeffrey@mycloud.org/urn:schemas-upnp-org:device:
    MediaServer:4:uuid:e70e9d0e-d9eb-4748-b163-636a323e7950"
  type="result">
  <query
    xmlns="http://jabber.org/protocol/disco#info"/>
    . . .
    <identity category="pubsub" type="service"/>
    <feature var="http://jabber.org/protocol/pubsub"/>
    <feature var="http://upnp.org/protocol/cloud/1.0"/>
    . . .
  </query>
</iq>
```

### C.7.1.1 Verifying an existing UCCD [PubSub](#) Hierarchy

Upon confirmation of the [PubSub](#) service existence that supports UCA, as indicated above, a UCCD with evented state variables shall confirm there is an existing, current [PubSub collection](#) hierarchy for the specific UCCD by sending an <iq> stanza as described in the following template.

```
<iq
  id="vendor defined value"
  from="localpart@domainpart/resourcepart of UCCD conforming to
    section C.5.4"
  to="PubSubName of UCS supporting UCA (see footnote in section C.6.6.3)"
  type="get"
```

```
<query xmlns="http://jabber.org/protocol/disco#info"
  node="resourcepart"/>
</iq>
```

<iq>, iq@id, iq@from, iq@to. Required. See equivalent in previous template for <iq> attributes.

@type

Required. Shall be implemented according to [RFC-6120]. Shall have a value of "get".

<query>

Required. Shall have "http://jabber.org/protocol/disco#info" as the value for the xmlns attribute (see schema in [XEP-0060] section 5.3 Discover Node Information). Case sensitive.

@node

Required. Type is xsd:string. Shall have a value the same as resourcepart of the requesting UCCD.

A UCCD shall inspect the UCS response <iq> stanza for the inclusion of a <query:identity> sub-element associated with the resourcepart described above where the @category attribute shall have a value of "pubsub" and the @node attribute shall have a value of "collection". If the described matches are confirmed, the UCCD shall next confirm the current configuration (configIDCloud leaf node), otherwise, it shall create a collection node hierarchy as described in section C.7.1.

Upon confirmation of the existing UCCD level collection node, a UCCD shall confirm that the configured PubSub device collection node hierarchy configIDCloud item value matches the current UCCD configIDCloud element value by sending an <iq> stanza as described in the following template.

```
<iq
  id="vendor defined value"
  from="localpart@domainpart/resourcepart of UCCD conforming to
    section C.5.4"
  to="PubSubName of UCS supporting UCA (see footnote in section C.6.6.3)"
  type="get">
  <pubsub xmlns="http://jabber.org/protocol/pubsub">
    <items node="value of UCCD PubSub configIDCloud node"/>
  </pubsub>
</iq>
```

<iq>, iq@id, iq@from, iq@to. Required. See equivalent in previous template for <iq> attributes.

@type

Required. Shall be implemented according to [RFC-6120]. Shall have a value of "get".

<pubsub>

Required. Shall be implemented according to [XEP-0030].

Shall have "http://jabber.org/protocol/pubsub" as the value for the xmlns attribute. Case sensitive.

<items>

Required. Shall contain an <items> element with an @node attribute whose value is the expected, configured UCCD level collection node. Case sensitive.

The returned item(s) list should contain the configIDCloud leaf node and a collection node for each service identified in the UCCD DDD.



## C.7.2 Creating a UCCD PubSub collection

The generalized steps for creating a UCCD [PuBSub](#) hierarchy, when required, are:

A UCCD shall create a [PubSub node](#) of type [collection](#) with UCCD [resourcepart](#) as the [node](#) name as described in this section.

A UCCD shall create a configIdCloud [PubSub node](#) of type [leaf](#) with UCCD [resourcepart"/configIdCloud](#) as the [node](#) name and bind it to the UCCD [collection node](#) as described in this section.

For each service in the UCCD, the UCCD shall create a service [node](#) of type [collection](#) with [resourcepart"/servicename](#) as the [node](#) name and bind it to it to the UCCD [collection node](#) as described in this section.

For each evented state variable in a UCCD service, the device shall create a state variable event [node](#) of type [leaf resourcepart"/servicename/statevariable](#) as the [node](#) name and bind it to the corresponding service [collection node](#) as described in this section.

A UCCD, after creating the PubSub structure, shall publish an initial event item for all evented state variables as described in C.7.3.

After creating all the nodes as described above and publishing the initial event, the UCCD shall publish an [item](#) value of [resourcepart"/configIdCloud](#) [leaf node](#) with a value equivalent to the UCCD DDD [configIdCloud](#) element value as described in this section.

The UCCD shall create the [resourcepart](#), [resourcepart"/service](#), and [resourcepart"/service/statevariable](#) and [resourcepart"/configIdCloud](#) nodes by sending an [iq](#) stanza as described in the following template.

```
<iq
  id="vendor defined value"
  from="localpart@domainpart/resourcepart of UCCD conforming to
    section C.5.4"
  to="PubSubName of UCS supporting UCA (see footnote in section C.6.6.3)"
  type="set"
  <pubsub xmlns="http://jabber.org/protocol/pubsub"/>
    <create node="value of UCCD node name"/>
    <configure>
      <x xmlns="jabber:x:data" type="submit">
        <field var="FORM TYPE" type="hidden">
          <value>http://jabber.org/protocol/pubsub#node config</value>
        </field>
        <field var="pubsub#node type">
          <value>
            collection if device or service level node
            leaf if a configIdCloud or state variable event node
          </value>
        </field>
        <field var="pubsub#collection">
          <value>
            name of device collection node if this is a service node
            name of device collection node if this is a configIdCloud node
            name of service node if this is a evented state variable node
          </value>
        </field>
        <field var="pubsub#access model">
          <value> access model supported by UCS conforming to UCA</value>
        </field>
        <field var="pubsub#max items">
          <value>
            1 if the node is the configIdCloud leaf node,
            # if services in the device if the node is a device collection
            node,
```

```

# if evented state variables in the service if the node is a
# service collection node,
# 1 or more if the node is an evented state variable leaf node.
# The recommended value is 1 with a maximum of 10. DCPs with
# specific recommendations for UCA should define a value and
# corresponding reason for a value above 1.
    </value>
  </field var="pubsub#other">
    </value>other</value>
  </field>
</x>
</configure>
</pubsub>
</iq>

```

<iq>, iq@id, iq@from, iq@to. Required. See equivalent in previous template for <iq> attributes.

@type

Required. Shall be implemented according to [RFC-6120]. Shall have a value of "set".

<pubsub>

Required. Shall have "<http://jabber.org/protocol/pubsub>" as the value for the xmlns attribute (see schema in [XEP-0060]). Case sensitive.

<create>

Required. Shall be implemented according to [XEP-0060].

Shall have a value of either resourcepart of the UCCD JID, or the concatenation of UCCD JID resourcepart with "/" with the value of serviceType element or the concatenation of UCCD JID resourcepart with "/" with the value of the serviceType element with "/" with the value of the stateVariable element dependent on whether the UCCD is creating the UCCD deviceType collection node, the UCCD serviceType collection node or the UCCD stateVariable leaf node respectively.

Note that if the UCCD is creating a stateVariable leaf node that the combination of the serviceType value and the stateVariable value shall only be for state variables that are valid for the service indicated and shall only be created for those state variables that have an @sendEvents attribute value of "yes".

Note that the PubSub nodes should be created in the order in which they appear in the related device description response in section C.6.6.2.

<configure>

Required. Shall have "<http://jabber.org/protocol/pubsub>" as the value for the xmlns attribute (see schema in [XEP-0060] section XML Schemas). Case sensitive.

<field>

Required. Shall be implemented according to [XEP-0060].

Shall contain the x, pubsub#node type, pubsub#collection, pubsub#access model, and pubsub#max items as described in the template above.

A UCS shall support the "whitelist" access model.

In the example below, a UCCD creates its initial eventing PubSub structure.

```

UCCD:jeffrey@mycloud.org/urn:schemas-upnp-
org:device:MediaServer:4:uuid:e70e9d0e-d9eb-4748-b163-636a323e7950
<iq
  id="9b6LhcCnH2AtIjtXixM7"

```

```
to="pubsub.mycloud.org"
type="set">
<pubsub xmlns="http://jabber.org/protocol/pubsub">
  <create node="urn:schemas-upnp-org:device:
    MediaServer:4:uuid:e70e9d0e-d9eb-4748-b163-636a323e7950"/>
  <configure>
    <x xmlns="jabber:x:data" type="submit">
      <field var="FORM_TYPE" type="hidden">
        <value>http://jabber.org/protocol/pubsub#node_config</value>
      </field>
      <field var="pubsub#node_type">
        <value>collection</value>
      </field>
    </x>
  </configure>
</pubsub>
</iq>
```

UCCD:jeffrey@mycloud.org/urn:schemas-upnp-  
org:device:MediaServer:4:uuid:e70e9d0e-d9eb-4748-b163-636a323e7950

```
<iq
  id="ztIyU6LYtLPUu9yQslq2"
  to="PubSubName of UCS"
  type="set">
<pubsub xmlns="http://jabber.org/protocol/pubsub"/>
  <create node="urn:schemas-upnp-org:device:
    MediaServer:4:uuid:e70e9d0e-d9eb-4748-b163-
    636a323e7950/ContentDirectory"/>
  <configure>
    <x xmlns="jabber:x:data" type="submit">
      <field var="FORM_TYPE" type="hidden">
        <value>http://jabber.org/protocol/pubsub#node_config</value>
      </field>
      <field var="pubsub#node_type">
        <value>collection</value>
      </field>
    </x>
  </configure>
</pubsub>
</iq>
```

UCCD:jeffrey@mycloud.org/urn:schemas-upnp-  
org:device:MediaServer:4:uuid:e70e9d0e-d9eb-4748-b163-636a323e7950

```
<iq
  id="I6nvuRN10Ac2XAdYiPeT"
  to="pubsub.mycloud.org"
  type="set">
<pubsub xmlns="http://jabber.org/protocol/pubsub"/>
  <create node="urn:schemas-upnp-org:device:
    MediaServer:4:uuid:e70e9d0e-d9eb-4748-b163-
    636a323e7950/ContentDirectory/SystemUpdateID"/>
  <configure>
    <x xmlns="jabber:x:data" type="submit">
      <field var="FORM_TYPE" type="hidden">
        <value>http://jabber.org/protocol/pubsub#node_config</value>
      </field>
      <field var="pubsub#node_type">
        <value>leaf</value>
      </field>
    </x>
  </configure>
</pubsub>
</iq>
```

UCCD:jeffrey@mycloud.org/urn:schemas-upnp-org:device:MediaServer:4:uuid:e70e9d0e-d9eb-4748-b163-636a323e7950

```
<iq
  id="bUv256CyvkxyKKyVCQ2h"
  to="pubsub.mycloud.org"
  type="set"
  <pubsub xmlns="http://jabber.org/protocol/pubsub"/>
    <create node="urn:schemas-upnp-org:device:
      MediaServer:4:uuid:e70e9d0e-d9eb-4748-b163-
      636a323e7950/ContentDirectory/LastChange"/>
    <configure>
      <x xmlns="jabber:x:data" type="submit">
        <field var="FORM_TYPE" type="hidden">
          <value>http://jabber.org/protocol/pubsub#node_config</value>
        </field>
        <field var="pubsub#node_type">
          <value>leaf</value>
        </field>
      </x>
    </configure>
  </pubsub>
</iq>
```

Once all **collection nodes** (device and service) and all **leaf nodes** (configIdCloud and evented state variables) have been created, and all current events published as items to their respective PubSub nodes (see section C.7.3), the UCCD shall update the configIdCloud **leaf node** with the current value of [configIdCloud](#) (see C.6.1) of the configuring UCCD as described in the following template.

```
<iq
  id="vendor defined value"
  from="localpart@domainpart/resourcepart of UCCD conforming to
    section C.5.4"
  to="PubSubName of UCS supporting UCA (see footnote in section C.6.6.3)"
  type="set">
  <pubsub xmlns="http://jabber.org/protocol/pubsub" />
    <publish node="name of configIdCloud leaf node as described above">
      <item>
        <e:configIdCloud xmlns:e="urn:schemas-upnp-org:cloud-1-0">
          value of configIdCloud
        </e:configIdCloud>
      </item>
    </publish>
  </pubsub>
</iq>
```

<iq>, iq@id, iq@from, iq@to. Required. See equivalent in previous template for <iq> attributes.

@type

Required. Shall be implemented according to [RFC-6120]. Shall have a value of "set".

<pubsub>

Required. Shall be implemented according to [XEP-0060] for a pubsub element. Shall have "http://jabber.org/protocol/pubsub" as the value for the xmlns attribute (see schema in [XEP-0060]). Case sensitive. Shall contain a publish element whose @node attribute is the configIdCloud node. Shall contain an item element containing e:configIdCloud element according to urn:schemas-upnp-org:cloud-1-0.

The UCCD shall monitor the response to the node creation stanzas to make sure that each node is created successfully. If a node creation fails the UCCD should re-try the creation. The UCCD shall not publish a configIDCloud leaf node (other than with a value of "CONFIGURING") until it has confirmed that the complete PubSub event structure for the current UCCD has been configured.

A UCCD shall be parsimonious with any updates to its PubSub collection to avoid unnecessary stanza traffic as any node or item deletion will result in notifications to all subscribed UCC-CPs. Therefore, a UCCD should only remove collection or event node(s) that are no longer part of the current configuration. For example, a UCCD should not remove the ContentDirectory service collection node from a MediaServer collection node since it is required for all MediaServers or a SystemUpdateID state variable event node since it is required for all ContentDirectory services.

PubSub node and item deletion is not described explicitly in the UCA but all UCCDs and UCSs shall support node and item deletion as described in [XEP-0060] sections "Delete a Node" and "Delete and Item from a Node" respectively; and shall use the to and from stanza addressing as described previously in this section.

In a newer instance of the MediaServer, support for the LastChange state variable is dropped. After first publishing a "CONFIGURING" state to the configIDCloud leaf node, the UCCD deletes the state variable leaf node "LastChange" as follows:

```
UCCD:jeffrey@mycloud.org/urn:schemas-upnp-org:device:MediaServer:4:uuid:e70e9d0e-d9eb-4748-b163-636a323e7950
```

```
<iq
  id="4bzS1rcVVWtw7koYAhVb"
  to="pubsub.mycloud.org"
  type="set"
  <pubsub xmlns="http://jabber.org/protocol/pubsub/owner"/>
    <delete node="urn:schemas-upnp-org:device:
      MediaServer:4:uuid:e70e9d0e-d9eb-4748-b163-
      636a323e7950/ContentDirectory/LastChange"/>
  </pubsub>
</iq>
```

Once the UCCD event structure is updated the UCCD also updates the configIDCloud leaf node with the new value. Note that the UCCD will have already updated its presence with the new configIDCloud value.

### C.7.3 Publishing a UCCD PubSub event

When a UCCD events a state variable, it shall update the corresponding PubSub leaf node by sending an <iq> stanza to its UCC PubSub service containing the publish element as described in the following template.

```
<iq
  id="vendor defined value"
  from="localpart@domainpart/resourcepart of UCCD conforming to
    section C.5.4"
  to="PubSubName of UCS supporting UCA (see footnote in section C.6.6.3)"
  type="set">
  <pubsub xmlns="http://jabber.org/protocol/pubsub"/>
    <publish node="name of event leaf node as described above">
      <item>
        <e:propertyset xmlns:e="urn:schemas-upnp-org:event-1-0">
          <e:property>
            <variableName>new value</variableName>
          </e:property>
        </e:propertyset>
      </item>
    </publish>
  </pubsub>
</iq>
```

```

        </e:propertyset>
      </item>
    </publish>
  </pubsub>
</iq>

```

<iq>, iq@id, iq@from, iq@to. Required. See equivalent in previous template for <iq> attributes.

@type

Required. Shall be implemented according to [XEP-0060]. Shall have a value of "set".

<pubsub>

Required. Shall be implemented according to [XEP-0060] for a pubsub element. Shall have "http://jabber.org/protocol/pubsub" as the value for the xmlns attribute (see schema in [XEP-0060]). Case sensitive.

Shall contain a publish element whose @node attribute is the name of the event node.

Shall contain an item element containing an e:propertyset element as described in section 4.3.2. Shall contain only <variableName> elements corresponding to the state variable associated with the publish node name.

In the example below a UCCD (MediaServer) publishes and update to the ContentDirectory service SystemUpdateID state variable.

UCCD:jeffrey@mycloud.org/urn:schemas-upnp-org:device:MediaServer:4:uuid:e70e9d0e-d9eb-4748-b163-636a323e7950

```

<iq
  id="WEnCONy2g1rJQ9SN0TWN"
  to="pubsub.mycloud.org"
  type="set">
  <pubsub xmlns="http://jabber.org/protocol/pubsub"/>
    <publish node="urn:schemas-upnp-org:device:
      MediaServer:4:uuid:e70e9d0e-d9eb-4748-b163-636a323e7950
      /ContentDirectory/SystemUpdateID">
      <item>
        <e:propertyset xmlns:e="urn:schemas-upnp-org:event-1-0">
          <e:property>
            <SystemUpdateID>2716658</SystemUpdateID>
          </e:property>
        </e:propertyset>
      </item>
    </publish>
  </pubsub>
</iq>

```

Upon successful publication of the event, the UCCD shall receive an <iq> stanza identifying the specific unique ID of the published event as described in the following template.

```

<iq
  id="vendor defined value"
  to="localpart@domainpart/resourcepart of UCCD conforming to
    section C.5.4"
  from="PubSubName of UCS supporting UCA (see footnote in section C.6.6.3)"
  type="set">
  <pubsub xmlns="http://jabber.org/protocol/pubsub"/>
    <publish node="name of event leaf node as described above">
      <item id="unique id of published leaf node event"/>
    </publish>
  </pubsub>
</iq>

```

<iq>, iq@id, iq@from, iq@to. Required. See equivalent in previous template for <iq> attributes.

@type

Required. Shall be implemented according to [RFC-6120]. Shall have a value of "[result](#)".

<pubsub>

Required. Shall be implemented according to [XEP-0060] for a [pubsub](#) element. Shall have "<http://jabber.org/protocol/pubsub>" as the value for the xmlns attribute (see schema in [XEP-0060]). Case sensitive. Shall contain a [publish](#) element whose @[node](#) attribute is the name of the event [node](#). Should contain an [item](#) element whose @[id](#) attribute contains a unique value identifying the specific publication.


If the publication fails then the subscribing UCCD should receive a <[iq](#)> stanza with the proper error code according to [XEP-0060].

Below is an example of the acknowledgement the UCCD receives from the event publication above.

```
P->UCCD:jeffrey@mycloud.org/urn:schemas-upnp-org:device:MediaServer:4:uuid:e70e9d0e-d9eb-4748-b163-636a323e7950
```

```
<iq
  id="WEnCONy2g1rJQ9SN0TWN"
  to="pubsub.mycloud.org"
  type="set">
  <pubsub xmlns="http://jabber.org/protocol/pubsub"/>
    <publish node="urn:schemas-upnp-org:device:MediaServer:4:uuid:e70e9d0e-d9eb-4748-b163-636a323e7950/ContentDirectory/SystemUpdateID">
      <item id="ae890ac52d0df67ed7cfd51b644e901"/>
    </publish>
  </pubsub>
</iq>
```

Note that some additional considerations for special case events may be needed, such as a ContentDirectory service [LastChange](#) event; in these cases, a specific UCA Annex will be created to describe the support for the particular DCP.

	<p>Typically the <a href="#">PubSub</a> event <a href="#">leaf node</a> buffer has a depth of 1, therefore it is recommended that a <a href="#">PubSub retract</a> message not be sent (to save on bandwidth) unless an event buffer specifically defined to be of length greater than 1 has been configured.</p>
---	---

UCC-CPs subscribed to the event ([leaf node](#)) or any corresponding [collection node](#) will receive the event with item id and possible payload as described in the following template depending whether the UCCD configures with or without [#deliver payloads](#) set and the [item node](#) is a [leaf](#) or [collection](#) type (see section 7 of [XEP-0060] for more detail).

```
<message
  id="unique id assigned by PubSubName of UCS supporting UCA "
  to="localpart@domainpart/resourcepart of subscribed UCC-CP conforming to section C.5.4"
  from="PubSubName of UCS supporting UCA (see footnote in section C.6.6.3)">
  <event xmlns="http://jabber.org/protocol/pubsub#event"/>
    <items node="name of event node as described above">
      <item id="unique id assigned by PubSubName of UCS supporting UCA">
        <e:propertyset xmlns:e="urn:schemas-upnp-org:event-1-0">
          <e:property>
            <variableName>new value</variableName>
          </e:property>
        </e:propertyset>
      </item>
    </items node>
  </event>
</message>
```

```
</items>
</event>
</message>
```

<message>

@id

Required. Shall be implemented according to [RFC-6120]. Case sensitive.

@from

Required. Shall be implemented according to [RFC-6120]. Shall have a value of the JID of the PubSub service publishing the event. Case sensitive.

@to.

Required. Shall be implemented according to [RFC-6120]. Shall have a value of the [full JID](#) of the UCC-CP subscribed to the event. Case sensitive.

<event>

Required. Shall be implemented according to [XEP-0060] for a [event](#) element. Shall have "<http://jabber.org/protocol/pubsub#event>" as the value for the xmlns attribute (see schema in [XEP-0060]). Case sensitive.

<items>

Required. Shall be implemented according to [XEP-0060] for an [items](#) element. Case sensitive.

@node

Required. Shall be implemented according to [XEP-0060] and contain a value corresponding to the event node.

<item>

Required. Shall be implemented according to [XEP-0060]. Case sensitive.

May contain an XML element whose value corresponds to the event payload, that is, an [e:propertyset](#) element as described in section 4.3.2.

@id

Required. Shall be implemented according to [XEP-0060]. Case sensitive.

In the example below, a UCC-CP subscribed to the event for the MediaServer in the previous example, receives notification of ContentDirectory service SystemUpdateID state variable change, and in this case, with delivery of the payload.

```
P->UCC-CP: jeffrey@mycloud.org/urn:schemas-upnp-org:cloud-1-0:
ControlPoint:1:ad93e8f5-634b-4123-80ca-225886a5c0e8
```

```
<message
  id="2g1rJQWEnCONy9SN0TWN"
  from="pubsub.mycloud.org"
  to="jeffrey@mycloud.org/urn:schemas-upnp-org:cloud-1-0:
    ControlPoint:1:ad93e8f5-634b-4123-80ca-225886a5c0e8">
  <event xmlns="http://jabber.org/protocol/pubsub#event"/>
    <items node="urn:schemas-upnp-org:device:
      MediaServer:4:uuid:e70e9d0e-d9eb-4748-b163-636a323e7950
      /ContentDirectory/SystemUpdateId">
      <item id="ae890ac52d0df67ed7cfd51b644e901">
        <e:propertyset xmlns:e="urn:schemas-upnp-org:event-1-0">
          <e:property>
            <SystemUpdateID>2716658</SystemUpdateID>
          </e:property>
        </e:propertyset>
```



```

    </item>
  </publish>
</pubsub>
</iq>

```

#### C.7.4 Subscribing to a UCCD PubSub collection

When subscribing to a UCCD, configIDCloud or evented state variable [leaf node](#) or a device or service [collection node](#) on a UCCD [PubSub](#), a UCC-CP shall send an [iq](#) stanza to the UCCDs [PubSub](#) service as described in the following template.

```

<iq
  id="vendor defined value"
  from="localpart@domainpart/resourcepart of subscribing UCC-CP conforming
        to section C.5.4"
  to=" PubSubName of UCS supporting UCA (see footnote in section C.6.6.3)"
  type="set">
  <pubsub xmlns="http://jabber.org/protocol/pubsub"/>
    <subscribe
      node="name of event leaf node
            or device collection node
            or service collection node
            as described above that the UCC-CP is subscribing to"
      jid="full JID of subscribing UCC-CP"/>
    </pubsub>
  </iq>

```

[iq](#), [iq@id](#), [iq@from](#), [iq@to](#). Required. See equivalent in previous template for [iq](#) attributes.

[@type](#)

Required. Shall be implemented according to [RFC-6120]. Shall have a value of ["set"](#).

[pubsub](#)

Required. Shall be implemented according to [XEP-0060] for a ["subscribe"](#) element.

Shall have ["http://jabber.org/protocol/pubsub"](#) as the value for the [xmlns](#) attribute (see schema in [XEP-0060]). Case sensitive.

Shall contain a [subscribe](#) node as defined in [XEP-0060].

Shall contain a [@jid](#) attribute whose value is equivalent to that of the UCC-CP subscribing to a [pubsub item](#).

Shall contain a [@node](#) attribute whose value is equivalent to the [leaf node](#) name of a state variable, configIDCloud, icon, or related parent event [collection node](#) for which the UCC-CP is subscribing.

In the example below, a UCC-CP subscribes to the ContentDirectory service [collection node](#).

UCC-CP:jeffrey@mycloud.org/urn:schemas-upnp-org:cloud-1-0:ControlPoint:1:ad93e8f5-634b-4123-80ca-225886a5c0e8

```

<iq
  id="AbaDM3Au4atsk8kXsyHxYbMh"
  to="pubsub.mycloud.org"
  type="set">
  <pubsub xmlns="http://jabber.org/protocol/pubsub"/>
    <subscribe
      node="urn:schemas-upnp-org:device:
            MediaServer:4:uuid:e70e9d0e-d9eb-4748-b163-636a323e7950/
            ContentDirectory"
      jid="jeffrey@mycloud.org/urn:schemas-upnp-org:cloud-1-0:

```

```
ControlPoint:1:ad93e8f5-634b-4123-80ca-225886a5c0e8"/>
</pubsub>
</iq>
```

Upon a successful subscribe the UCC-CP should expect to receive an `<iq>` stanza of type `"result"` containing the elements as described in the following template.

```
<iq
  from="localpart@domainpart/resourcepart of subscribing UCC-CP conforming
        to section C.5.4"
  to=" PubSubName of UCS supporting UCA (see footnote in section C.6.6.3)"
  type="result">
  <pubsub xmlns="http://jabber.org/protocol/pubsub">
    <subscription
      node="name of event leaf node
            or device collection node
            or service collection node
            as described above that the UCC-CP is subscribing to"
      jid="full JID of subscribing UCC-CP"
      subid="subscription unique identifier generated by PubSub service"
      subscription="subscribed"/>
    </pubsub>
  </iq>
```

`<iq>`, `iq@id`, `iq@from`, `iq@to`. Required. See equivalent in previous template for `<iq>` attributes.

`@type`

Required. Shall be implemented according to [RFC-6120]. Shall have a value of `"result"`.

`<pubsub>`

Required. Shall be implemented according to [XEP-0060] for a `subscription` element.

Shall contain a `@jid` attribute whose value is equivalent to the UCC-CP that has successfully subscribed to a `pubsub item`.

Shall contain a `@node` attribute whose value is equivalent to the `leaf node` name of the state variable, `configIdCloud`, `icon`, or related parent event `collection node` for which the UCC-CP has a subscription.

Allowed to contain a `@subid` attribute whose value is a unique value describing the specific subscription (similar to unique subscription identifier as described in section C.7.1.1).

Shall contain a `@subscription` attribute whose value is `"subscribed"`.

Continued from the previous example, the UCC-CP receives success from the subscribe to the ContentDirectory service `collection node`.

```
P->UCC-CP:jeffrey@mycloud.org/urn:schemas-upnp-org:cloud-1-0:ControlPoint:1:ad93e8f5-634b-4123-80ca-225886a5c0e8
<iq
  id="AbaDM3Au4atsk8kXsyHxYbMh"
  to="jeffrey@mycloud.org/urn:schemas-upnp-org:cloud-1-0:
      ControlPoint:1:ad93e8f5-634b-4123-80ca-225886a5c0e8"
  type="result">
  <pubsub xmlns="http://jabber.org/protocol/pubsub">
    <subscription
      node="urn:schemas-upnp-org:device:
            MediaServer:4:uuid:e70e9d0e-d9eb-4748-b163-636a323e7950/
            ContentDirectory"
      jid="jeffrey@mycloud.org/urn:schemas-upnp-org:cloud-1-0:
            ControlPoint:1:ad93e8f5-634b-4123-80ca-225886a5c0e8"
      subid="2deed9ef1d677508ae63e12f3beb9b6006eb69c8"
      subscription="subscribed"/>
    </pubsub>
```

</iq>

If the subscribe fails then the subscribing UCC-CP should receive a <iq> stanza with the proper error code according to [XEP-0060].

All UCC-CP subscriptions shall be of the type **full JID**, although **bare JID** subscriptions are allowed under XMPP as described below.

From [XEP-0060] Section 6.1.6 Multiple Subscriptions

When the **PubSub** service generates event notifications, it should send only one event notification to an entity that has multiple subscriptions, rather than one event notification for each subscription. By "entity" here is meant the **JID** specified for the subscription, whether **bare JID** or **full JID**; however, if the same **bare JID** has multiple subscriptions but those subscriptions are for different **full JIDs** (e.g., one subscription for user@domain.tld/foo and another subscription for user@domain.tld/bar), the service shall treat those as separate **JIDs** for the purpose of generating event notifications.

### C.7.5 Unsubscribing to a UCCD **PubSub** collection

When un-subscribing to a UCCD evented state variables **leaf node** or a device or service **collection node** on a UCCD cloud interface (**PubSub**), a UCC-CP shall send an <iq> stanza to the UCCDs **PubSub** service containing the **unsubscribe** element as described in the following template.

```
<iq
  id="vendor defined value"
  from="localpart@domainpart/resourcepart of subscribed UCC-CP conforming
        to section C.5.4"
  to="PubSubName of UCS supporting UCA (see footnote in section C.6.6.3)"
  type="set">
  <pubsub xmlns="http://jabber.org/protocol/pubsub"/>
    <unsubscribe
      node="name of event leaf node as described above that
            the UCC-CP is unsubscribing from"
      jid="full JID of subscribed UCC-CP"/>
      <subid="subscription unique identifier generated by PubSub service"
    </pubsub>
</iq>
```

<iq>, iq@id, iq@from, iq@to. Required. See equivalent in previous template for <iq> attributes.

@type

Required. Shall be implemented according to [RFC-6120]. Shall have a value of **"set"**.

<pubsub>

Required. Shall be implemented according to [XEP-0060] for an **unsubscribe** element.

Shall contain a @**jid** attribute whose value is equivalent to the UCC-CP that is unsubscribing from **pubsub node**.

Shall contain a @**node** attribute whose value is equivalent to the **leaf node** name of the state variable, configIdCloud, icon, or related parent **collection node** for which the UCC-CP is unsubscribing.

Allowed to contain a @**subid** attribute whose value is a unique value describing the specific subscription (similar to unique subscription identifier as described in section C.7.1.1).

Continued from the previous example, the UCC-CP unsubscribes to the ContentDirectory service collection node.

```
P->UCC-CP:jeffrey@mycloud.org/urn:schemas-upnp-org:cloud-1-0:ControlPoint:1:ad93e8f5-634b-4123-80ca-225886a5c0e8
```

```
<iq
  id="cAdMHPtF2aMZqdrkQV82CK5Z"
  to="jeffrey@mycloud.org/urn:schemas-upnp-org:cloud-1-0:
    ControlPoint:1:ad93e8f5-634b-4123-80ca-225886a5c0e8"
  type="set">
  <pubsub xmlns="http://jabber.org/protocol/pubsub"/>
    <unsubscribe
      node="urn:schemas-upnp-org:device:
        MediaServer:4:uuid:e70e9d0e-d9eb-4748-b163-636a323e7950/
        ContentDirectory"
      jid="jeffrey@mycloud.org/urn:schemas-upnp-org:cloud-1-0:
        ControlPoint:1:ad93e8f5-634b-4123-80ca-225886a5c0e8"
      subid="2deed9ef1d677508ae63e12f3beb9b6006eb69c8"/>
    </pubsub>
  </iq>
```

Upon a successful unsubscribe the UCC-CP should expect to receive an <iq> stanza of type "result" containing an empty body as described in the following template.

```
<iq
  id="vendor defined value"
  from="localpart@domainpart/resourcepart of subscribed UCC-CP conforming
    to section C.5.4"
  to="PubSubName of subscribed UCCD UCS PubSub service"
  type="result"/>
```

<iq>, iq@id, iq@from, iq@to. Required. See equivalent in previous template for <iq> attributes.

@type

Required. Shall be implemented according to [RFC-6120]. Shall have a value of "result".

From [XEP-0060] Section 6.1.6 Multiple Subscriptions

When the **PubSub** service generates event notifications, it should send only one event notification to an entity that has multiple subscriptions, rather than one event notification for each subscription. By "entity" here is meant the **JID** specified for the subscription, whether **bare JID** or **full JID**; however, if the same **bare JID** has multiple subscriptions but those subscriptions are for different **full JIDs** (e.g., one subscription for user@domain.tld/foo and another subscription for user@domain.tld/bar), the service shall treat those as separate **JIDs** for the purpose of generating event notifications.

If the unsubscribe fails then the unsubscribing UCC-CP should receive a <iq> stanza with the proper error code according to [XEP-0060].

### C.7.6 Permissions model

Principals of UCA PubSub permission models.

- Personal Eventing Protocol is the preferred configuration.
- PubSub leaf and collection node ownership should be restricted to the full JID to prevent cross posting of events between UCCDs and UCC-CPs or non-UCA JIDs.
- The access model for UCCD event collection and leaf nodes should be of type "roster".

- All UCCDs and UCC-CPs should be in the base `<group/>` named "[UPnPCloud](#)".
- All UCCDs and UCC-CPs in the "[UPnPCloud](#)" `roster <group/>` should have [affiliations](#) with the event [collection](#) and [leaf node](#)(s) associated with their [bare JID](#) so that any event subscription can be auto accepted.

## C.8 SOAP over XMPP (Analog of Control)

Like UDA, UCA uses SOAP for control and specifically extends the SOAP over XMPP as described in [XEP-0072]<sup>24</sup>.

A UCCD shall be capable of receiving and responding to an invoked action using SOAP over XMPP as defined in [XEP-0072] and this specification.

A UCC-CP shall be capable of invoking an Action using SOAP over XMPP as defined in [XEP-0072] and this specification.

To verify SOAP support at the XMPP level a UCC-CP can send an `<iq>` stanza containing an `<query xmlns=http://jabber.org/protocol/disco#info>` element to a UCCD. If such a stanza is received by a UCCD and no error conditions are encountered the UCCD should include in its response `<query>` element sub-elements `<identity category="automation" type="soap">` and `<feature var=http://jabber.org/protocol/soap>` element.

An example is shown below in the case where [disco#info](#) is used to indicate SOAP support.

```
UCC-CP:jeffrey@mycloud.org/urn:schemas-upnp-org:ControlPoint:1:ad93e8f5-634b-4123-80ca-225886a5c0e8->UCCD
```

```
<iq
  to="jeffrey@mycloud.org/urn:schemas-upnp-org:device:MediaServer:4:
    uuid:e70e9d0e-d9eb-4748-b163-636a323e7950"/>
  id="cp-1-do-you-soap"
  type="get">
  <query xmlns="http://jabber.org/protocol/disco#info"/>
</iq>
```

```
UCCD:jeffrey@mycloud.org/urn:schemas-upnp-
org:device:MediaServer:4:uuid:e70e9d0e-d9eb-4748-b163-636a323e7950->UCC-CP
```

```
<iq
  to="jeffrey@mycloud.org/urn:schemas-upnp-org:ControlPoint:1:
    ad93e8f5-634b-4123-80ca-225886a5c0e8"/>
  id="cp-1-do-you-soap"
  type="result">
  <query xmlns="http://jabber.org/protocol/disco#info"/>
    <identity category="automation" type="soap"/>
    <feature var="http://jabber.org/protocol/soap"/>
  </query>
</iq>
```

SOAP messages are exchanged using `<iq>` stanzas where the invoking UCC-CP sends an `<iq>` stanza with the `iq@type` attribute value of "`set`" with the service `serviceId` contained in the header and the SOAP action contained in the body of the `<iq>` stanza as follows:

To control a UCCD, a UCC-CP shall send an `<iq>` stanza as described in the following template.

---

<sup>24</sup> Note that even though [XEP-0072] is defined for SOAP 1.2 only SOAP 1.1 is required for UCA, for example, SOAP 1.1 namespace is used as described in section 3.: Control.

```

<iq
  id="vendor defined value"
  from="localpart@domainpart/resourcepart of UCC-CP invoking action
        conforming to section C.5.4"
  to="localpart@domainpart/resourcepart of UCCD where action is invoked
        conforming to section C.5.4"
  type="set">
  <s:Envelope
    xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"
    s:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <s:Header mustUnderstand="1">
      <uc xmlns="urn:schemas-upnp-org:cloud-1-0" serviceId="serviceId"/>
    </s:Header>
    <s:Body>
      <u:actionName xmlns:u="urn:schemas-upnp-org:service:serviceType:v">
        <argumentName>in arg value</argumentName>
        <!-- other in args and their value go here, if any -->
      </u:actionName>
    </s:Body>
  </s:Envelope>
</iq>

```

<iq>, iq@id, iq@from, iq@to. Required. See equivalent in previous template for <iq> attributes.

@type

Required. Shall be implemented according to [RFC-6120]. Shall have a value of "set".

<s:envelope>

Required. Type is <XML>. Shall be implemented according to section 3.2 for invoking an Action.

<s:header>

Required. Type is <XML>. Shall be implemented as described in section 3.1.1.

@mustUnderstand

Required. Type is xsd:string. Shall have a value of "1".

<uc>

Required. Type is <XML>.

@xmlns. Required. xsd:string. Shall have a value of "[urn:schemas-upnp-org:cloud-1-0](http://schemas-upnp-org:cloud-1-0)".

@serviceId. Required. xsd:string. Shall have a value of "serviceId" corresponding to the service for with the action is invoked.

The UCCD shall inspect the serviceId included in the SOAP header field and attempt to execute the invoked Action on the matching service. When the Action is invoked successfully or with an Action error then the UCCD shall respond with an <iq> stanza whose iq@type attribute has a value of "result" and that conforms to one of the following templates for an Action response or error respectively.

Action successful response template is:

```

<iq
  id="vendor defined value"
  to="localpart@domainpart/resourcepart of UCC-CP invoking action
        conforming to section C.5.4"
  from="localpart@domainpart/resourcepart of UCCD where action is invoked
        conforming to section C.5.4"
  type="result">
  <s:Envelope

```

```

xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"
s:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
<s:Header mustUnderstand="1">
  <uc xmlns="urn:schemas-upnp-org:cloud-1-0" serviceld="serviceld"/>
</s:Header>
<s:Body>
  <u:actionNameResponse
    <argumentName>out arg value</argumentName>
    <!-- other in args and their value go here, if any -->
  </u:actionNameResponse>
</s:Body>
</s:Envelope>
</iq>

```

<iq>, iq@id, iq@from, iq@to. Required. See equivalent in previous template for <iq> attributes.

@type

Required. Shall be implemented according to [RFC-6120]. Shall have a value of "**result**" if the Action invocation is successful.

<s:envelope>

Required. Type is <XML>. Shall be implemented according to 3.2.2 for Action returning a success response.

Action error response template is:

```

<iq
  id="vendor defined value"
  to="localpart@domainpart/resourcepart of UCC-CP invoking action
    conforming to section C.5.4"
  from="localpart@domainpart/resourcepart of UCCD where action is invoked
    conforming to section C.5.4"
  type="result">
<s:Envelope
  xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"
  s:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
<s:Header mustUnderstand="1">
  <uc xmlns="urn:schemas-upnp-org:cloud-1-0" serviceld="serviceld"/>
</s:Header>
<s:Body>
  <s:Fault>
    <faultcode>s:Client</faultcode>
    <faultstring>UpnPError</faultstring>
    <detail>
      <UPnPError xmlns="urn:schemas-upnp-org:control-1-0">
        <errorCode>error_code</errorCode>
        <errorDescription>error_string</errorDescription>
      </UPnPError>
    </detail>
  </s:Fault>
</s:Body>
</s:Envelope>
</iq>

```

<iq>, iq@id, iq@from, iq@to. Required. See equivalent in previous template for <iq> attributes.

@type

Required. Shall be implemented according to [RFC-6120]. Shall have a value of "**result**" if the Action invocation generates a UDA error condition.

<s:envelope>

Required. Shall be implemented according to 3.2.5 returning an Action error response.

The following is an example of Action invocation with both a success or error response

UCC-CP:jeffrey@mycloud.org/urn:schemas-upnp-org:ControlPoint:1:ad93e8f5-634b-4123-80ca-225886a5c0e8

```
<iq
  to="jeffrey@mycloud.org/urn:schemas-upnp-org:device:MediaRenderer:3:
    uuid:88509d0e-e8f5-80ca-4123-225886a50ee7"
  id="cp-1-soap-action-1"
  type="set">
  <s:Envelope
    xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"
    s:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <s:Header mustUnderstand="1">
      <uc xmlns="urn:schemas-upnp-org:cloud-1-0"
        serviceId="RenderingControl"/>
    </s:Header>
    <s:Body>
      <u:SetVolume xmlns:u="urn:schemas-upnp-
org:service:RenderingControl:3">
        <DesiredVolume>20</DesiredVolume>
      </u:SetVolume>
    </s:Body>
  </s:Envelope>
</iq>
```

The UCCD responds with a SOAP action or SOAP error response as defined in 3.2

In this case below the Action is successful.

UCCD:jeffrey@mycloud.org/urn:schemas-upnp-org:device:MediaRenderer:3:uuid:88509d0e-e8f5-80ca-4123-225886a50ee7

```
<iq
  to="jeffrey@mycloud.org/urn:schemas-upnp-org:ControlPoint:1:
    ad93e8f5-634b-4123-80ca-225886a5c0e8"
  id="cp-1-soap-action-1"
  type="result">
  <s:Envelope
    xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"
    s:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <s:Header mustUnderstand="1">
      <uc xmlns="urn:schemas-upnp-org:cloud-1-0"
        serviceId="RenderingControl"/>
    </s:Header>
    <s:Body>
      <u:SetVolume xmlns:u="urn:schemas-upnp-org:
        service:RenderingControl:3">
        <DesiredVolume>20</DesiredVolume>
      </u:SetVolume>
    </s:Body>
  </s:Envelope>
</iq>
```

In the case below the Action requested was invalid and the UCCD returns an error 401 - "Action Invalid".

UCCD:jeffrey@mycloud.org/urn:schemas-upnp-org:device:MediaServer:4:uuid:e70e9d0e-d9eb-4748-b163-636a323e7950

```
<iq
  to="jeffrey@mycloud.org/urn:schemas-upnp-org:ControlPoint:1:
    ad93e8f5-634b-4123-80ca-225886a5c0e8"
```



```

id="cp-1-soap-action-1"
type="result">
<s:Envelope
  xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"
  s:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <s:Header mustUnderstand="1">
    <uc xmlns="urn:schemas-upnp-org:cloud-1-0"
      serviceId="RenderingControl"/>
  </s:Header>
  <s:Body>
    <s:Fault>
      <faultcode>s:Client</faultcode>
      <faultstring>UPnPError</faultstring>
      <detail>
        <UPnPError xmlns="urn:schemas-upnp-org:control-1-0">
          <errorCode>401</errorCode>
          <errorDescription>Invalid Action</errorDescription>
        </UPnPError>
      </detail>
    </s:Fault>
  </s:Body>
</s:Envelope>
</iq>

```

Note, the messages are sent through the user's UCS which is not shown in the exchanges.

## C.9 Support for Binary (Media) Transport

No specific support for the exchange of binary data is required at this time. Any additional transport (for example Audio-Video media) is left to the responsible UPnP Forum Working Committee or the vendor. It is highly recommended that UPnP Forum Working Committees publish a UCA Annex to their existing DCPs and SCPDs to integrate UCA into the existing specifications.



Exchange of Base64 encoded binary within XMPP stanzas as a regular means of transport is highly discouraged, as this will likely result in significant UCS slowdown and overall bad UCA experience for all UCCD and UCC-CPs.

## C.10 UCA errorCodes

The following errorCodes are similar in structure and schema to those in Table 3-3 but are not part of an Action error and instead are UCA specific errors returned as part of [<iq>](#) stanza response.

ErrorCode	errorDescription	Description
900	DeviceType mismatch	The resource part of the full JID of the "to" device does not match the DDD of the device.

## C.11 UCA Schemas

See B.6 for `urn:upnp-schemas-org:cloud-1-0.xsd`.

## C.12 Closing a UCA Session

Often a UCCD or UCC-CP will need to completely close its connection with its UCS. When doing so it will first issue a `<presence>` stanza of type "`unavailable`" and then close the XMPP session using the stream close elements as shown below.

Note that this closes both the directional streams: UCCD (or UCC-CP) to UCS and UCS to UCCD (or UCC-CP).

```
<presence type="unavailable">
```

UCCD (or UCC-CP) closes its stream to UCS.

```
UCCD (or UCC-CP) ->UCS
```

```
</stream:stream>
```

UCS closes its stream to UCCD (or UCC-CP)

```
UCS->UCCD (or UCC-CP)
```

```
</stream:stream>
```

Note, that until the `</stream:stream>` element is received by the UCS the UCS will consider the connection open for a limited period, even if the underlying TCP connection has been closed. When the XMPP clients have been idle or sleeping for a significant amount of time (as determined by the UCS; most likely 30 minutes) or have been disconnected the UCS will send a `<presence>` stanza of type "`unavailable`" on their behalf.

## C.13 UCA over BOSH and WebSocket

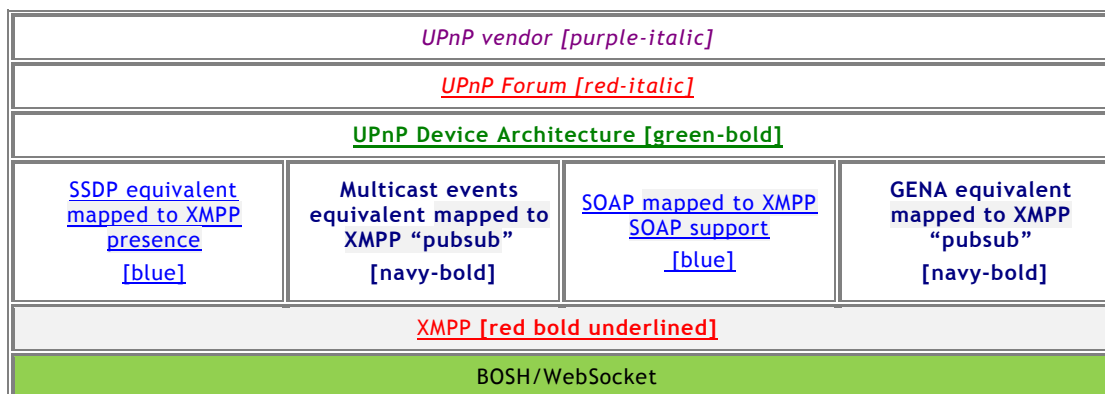
From [XEP-0124], BOSH "defines a transport protocol that emulates the semantics of a long-lived, bidirectional TCP connection between two entities (such as a client and a server) by efficiently using multiple synchronous HTTP request/response pairs without requiring the use of frequent polling or chunked responses; thus allowing a browser based UCCD or UCC-CP to connect with a Webserver based UCS. WebSocket [RFC-6455] offers additional advantages over BOSH as an alternative transport and its usage is likely to become more widespread in the future. Therefore, BOSH and WebSocket support for UCA is defined as follows:

A UCS shall support BOSH [XEP-0124].

A UCCD or UCC-CP which contains a web browser should support BOSH [XEP-0124].

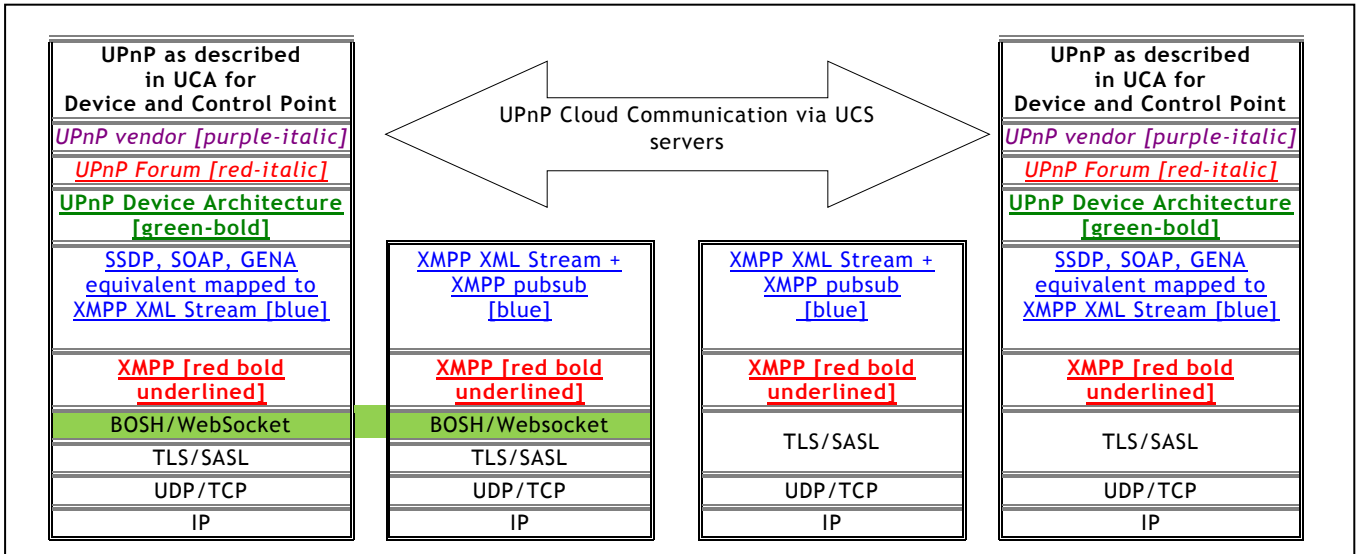
A UCS, UCCD or UCC-CP may support WebSocket [RFC-6455].

Figure C-11: — BOSH and WebSocket UCA Stack



TLS/SASL
TCP [black]
IP [black]

Figure C-12: — BOSH and WebSocket at UCA component stacks



UCCD/UCC-CP(A) → Server(A) → Server(B) → UCCD/UCC-CP(B)

