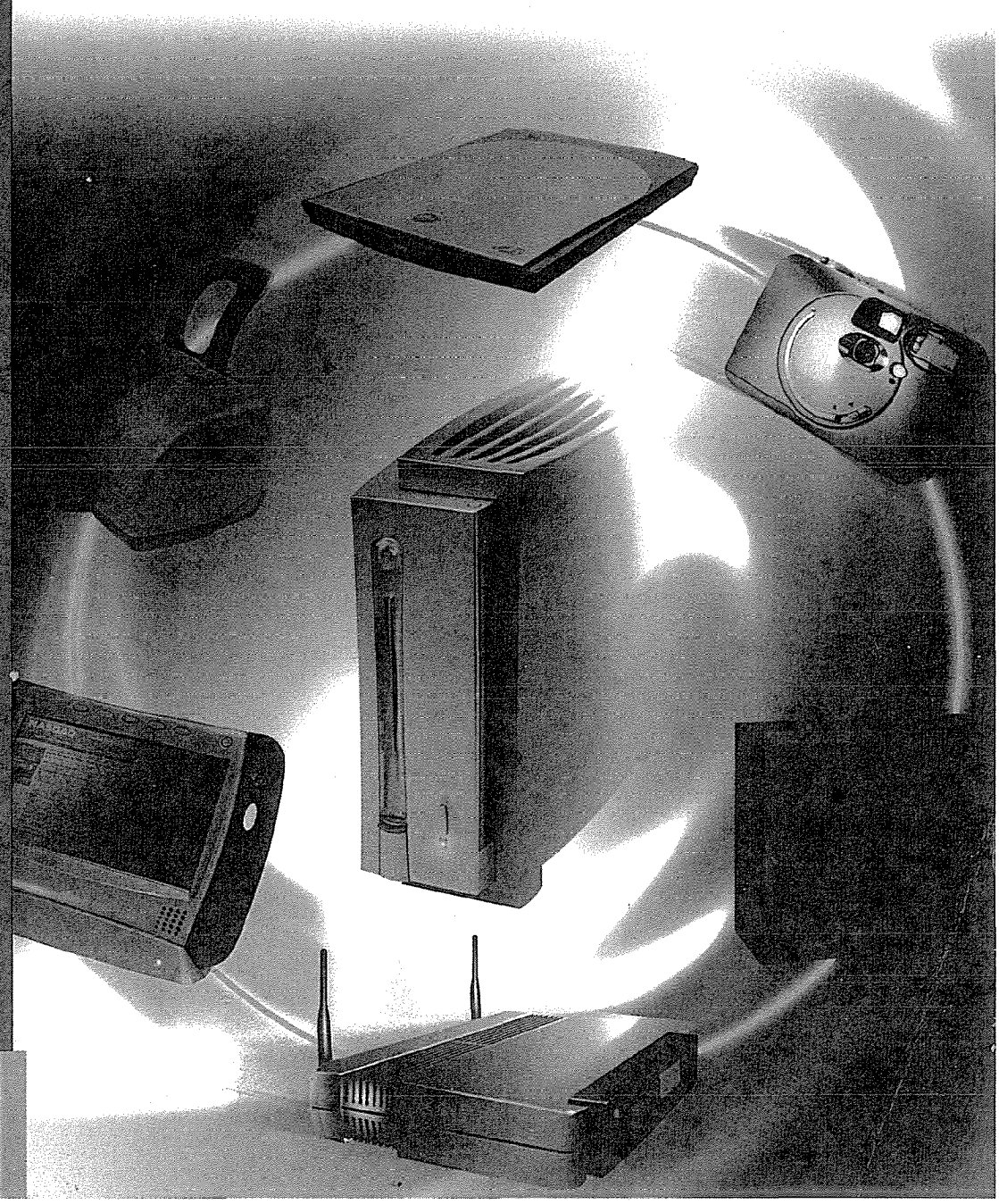
UPnP† Design by Example

A Software Developer's Guide to Universal Plug and Play

Michael Jeronimo and Jack Weast

INTEL PRESS

Books by Engineers, for Engineers



intel®

UPnP Design by Example

A Software Developer's Guide to Universal Plug and Play

Michael Jeronimo
Jack Weast

INTEL PRESS

ISBN 0-9717861-1-9

This book is printed on acid-free paper. ♾

Publisher: Richard Bowles

Editor: David J. Clark

Managing Editor: David B. Spencer

Content Manager: Stuart Goldstein

Text Design: Marianne Phelps

Composition: Octal Publishing, Incorporated

Graphic Art: Donna Lawless (illustrations), Ted Cyrek (cover)

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

First printing, April 2003

To Jenni. Thanks for believing in me. I appreciate your patience, support, encouragement, and sacrifice during the many months it took to write this book.

To Matty, Sean, and Portia. Thanks for keeping me smiling. You're the best bunch of heffalumps a dad could have. ☺

To Jan and John (Nana and Papa). Thanks for taking care of the kids during those Sunday afternoon writing sessions and for providing a home away from home.

—Michael

To Papa. The original Weast engineer who started it all; it is your life story that has taught me the value of hard work and dedication against even the greatest of odds.

To Aaron, whose natural ability to do everything better than your older brother continues to provide inspiration and drive for my own success.

—Jack

Part I

Introduction to the UPnP Architecture

Chapter 1

It Just Works

It's kind of fun to do the impossible.

—Walt Disney

People expect that when they bring a television or DVD player home, they can just plug it in, hook up a few cables, and the device will "just work." These devices perform their functions well and are easy for consumers to install. PC peripherals, on the other hand, have not been as easy to install. Users must be concerned with gory details such as device drivers to get devices to work properly. Recently, Universal Serial Bus (USB) and Plug-and-Play have improved the situation for PC peripherals so that devices can now be automatically detected and device drivers automatically installed. But networked devices, such as an Internet gateway or a networked printer, still require complicated manual setup and configuration.

The UPnP[1] standard brings the PC peripheral Plug-and-Play concept to the home network, with the same ease of use and automatic configuration that users have come to expect with Plug-and-Play devices. Just

[1] UPnP is a certification mark of the UPnP Implementers Corporation.

as devices can be plugged into the PC and automatically detected and configured, consumers of home networking equipment can now easily add UPnP devices to their home networks and have them just work.

Why the UPnP Standard?

When USB devices are plugged into a PC they are automatically detected by the operating system, which loads the appropriate software and makes the device available for applications to use. This automatic detection and configuration of devices makes it easy for the end user to add and use new devices.

Similar to a PC and its peripherals, there are various home networking devices, such as an Internet gateway or a networked printer, that the user may wish to connect to the local network. However, these devices usually require an administrator to configure them before they can be used. The difficulty of configuring home networking equipment has been a problem for consumers and a barrier to the adoption of home networking. With UPnP, users can add devices to the home network without installing drivers or configuring the devices before using them.

The Foundation for Home Networking

UPnP technology, along with other emerging technologies such as wireless networking and high-speed Internet connections, is transforming the home. Many devices, such as digital televisions and home audio equipment, are becoming UPnP technology-enabled. In time, other existing networks in the home, such as the power line, home entertainment, and telephone networks, will have bridging software that automatically makes devices on those networks appear as UPnP devices. The result will be a single, logical network of UPnP devices—a kind of "digital home platform" for entertainment, home automation, and other kinds of applications, as shown in Figure 1.1.

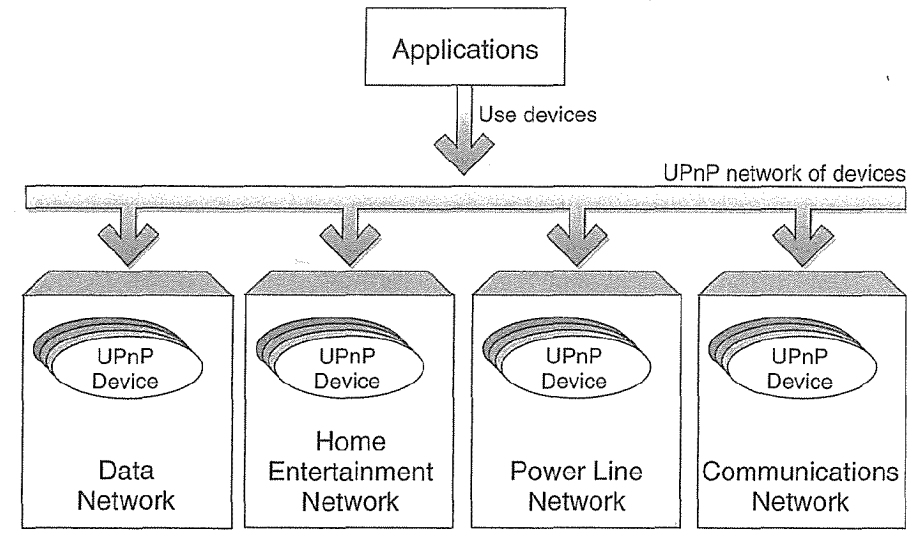


Figure 1.1 UPnP Technology Is the Foundation for Home Networking

What Is the UPnP Standard?

The UPnP architecture is designed to connect networked devices, such as PCs, entertainment equipment, and intelligent appliances. It defines a base set of standards for all devices to adhere to and conventions for describing devices and the services they provide.

The UPnP architecture leverages existing standards such as TCP/IP, HTTP, and XML instead of inventing new underlying mechanisms. The architecture consists of a set of standardized protocols that each UPnP technology-enabled device implements to provide for discovery, control, and data transfer between UPnP devices. UPnP technology can be supported on any common operating system or hardware platform, and it works with almost any type of physical networking media—wired or wireless—providing maximum user and developer choice.

The UPnP architecture provides:

- *Device Connectivity.* The UPnP architecture defines the protocols for devices to interact with other devices. UPnP devices can join and leave the network transparently, advertise their services, discover other devices and services, send events, and control other devices.

- *Ad-Hoc Networking.* UPnP devices can come together to form a network dynamically, without the need for dedicated networking infrastructure services, such as a server to manage address assignment. These *ad-hoc* networks are created on-the-fly and enable device connectivity without manual configuration.
- *Zero-Configuration Networks.* The UPnP architecture supports zero-configuration networking where the user is not required to configure devices before they are used on the network. The non-technical user will find it simple to add and use devices.
- *Standards-Based Architecture.* The UPnP architecture is based on open standards, including a foundation of existing and proposed standard Internet Engineering Task Force (IETF) and World Wide Web Consortium (W3C) protocols such as IP, TCP, UDP, HTTP, XML, and SOAP. Leveraging existing Internet-based technologies simplifies the design of UPnP devices.
- *Platform Independence.* The UPnP architecture is primarily a set of protocols and is not an API definition. The UPnP architecture keeps the implementation of the protocols private and does not require vendors to develop their implementations on any specific operating system, language, or hardware. With this approach, UPnP devices can be developed on any platform—a desirable trait in a network full of devices from many vendors, including consumer electronics companies.
- *Media and Device Independence.* UPnP technology can run on any medium for which there is an IP stack, including phone lines, power lines, Ethernet, RF, and IEEE 1394.
- *Programmatic and Manual Device Control.* The UPnP architecture enables applications to programmatically control home networking devices. In addition, users can manually control devices using the device's browser-based administrative interface.

User Scenarios

Many futuristic automation scenarios can be developed using UPnP devices. Here are a couple of examples that illustrate the power and flexibility of UPnP technology in the home of the future.

Watching a Movie

Arriving home after a long day at work, George decides to watch a movie. He happens to be in the kitchen getting a glass of juice from the fridge, so he calls up a list of recent movies on the screen near to him on the kitchen counter. George checks out some previews and then selects the movie to watch. The movie selection program turns on the home theater system and automatically starts the movie. The controlling program also dims the lights and adjusts the volume of the speakers. Settling in, George watches the movie for a while. Twenty minutes later, an alert pops up on the home theater screen indicating activity in the front yard. George puts the porch camera on the screen and sees the local pizza delivery man walking up the path to his door, delivering the pizza ordered earlier. George meets the delivery man at the door, takes the pizza, pays him, including a generous tip, and returns to the home theater room. Some time after the delivery man leaves, the kitchen lights and porch lights turn themselves off to conserve energy, having not detected any motion.

Home Maintenance

Every New Year's Day, Shannon does home maintenance. With her wireless PDA in hand, she walks through her house, examining the status of various systems and devices. The PDA displays the list of systems to be inspected in the house and tells Shannon what to look for, displaying instructions and pictures as needed.

Shannon starts in the garage with the water heater. She uses her PDA to view the operational parameters of the water heater. Using that information, she optimizes the heater's energy use, updating the heater's settings to monitor activity over a period of time and anticipate peak loads and off periods.

Shannon moves on to the kitchen where she uses the PDA to review the state of the dishwasher and the refrigerator. The dishwasher hasn't been working very well lately, so she uses the PDA to invoke the dishwasher's self-test. The test doesn't turn up any problems, so she calls up the dishwasher manual. The manual has a troubleshooting section that Shannon reads to find out what might be the problem. Shannon doesn't find any answers, so she uses the PDA to send an e-mail message to the manufacturer explaining the problem.

Shannon continues through the house, inspecting, calibrating, and making notes of things that she needs to buy for the house. She eventually completes this year's maintenance inspection and goes to the store to pick up the items she needs.

Key Themes

With a little playful daydreaming, you can probably envision many more scenarios like these that simplify life in the home, limited only by your imagination. Some key themes appear in many of the scenarios, such as automation, where devices automatically respond to events generated from other devices, and convenience, where the user is able to easily accomplish tasks. While the scenarios seem futuristic, one thing is certain—having a standard, open platform for home networking will inspire creativity. The UPnP standard is the underlying technology to help make scenarios like these real.

Let's take a step back from the future now and take a look at the UPnP Forum, the organization responsible for the UPnP standards.

The UPnP Forum

Microsoft Corporation introduced the UPnP initiative at the Consumer Electronics Show in January of 1999. The initiative was originally supported by companies such as Microsoft, Intel, Hewlett-Packard, Compaq, Dell, and many others, and was considered the next phase of the Plug-and-Play initiative introduced by Intel, Compaq, and Microsoft in 1992.

To guide the creation of the standards, a cross-industry group, the UPnP Forum, was created. Today, the Forum consists of more than 550 companies, including industry leaders in consumer electronics, computing, home automation, home security, appliances, printing, photography, computer networking, and mobile products.

The primary activities of the UPnP Forum include:

- Defining device standards based on the UPnP architecture
- Providing for the certification of devices
- Facilitating joint member promotion of UPnP

Device descriptions are XML documents, based on a device description document schema, that describe a particular kind of device. By defining and publishing UPnP device descriptions, members of the UPnP Forum create standard building blocks for home networking. The standards defined by the UPnP Forum are platform-neutral. Membership and participation in the design of device schema templates are open to any member companies. Companies interested in standardizing particular device classes are encouraged to join the UPnP Forum and participate in working committees to design schema templates for their devices.

Vendors can implement devices that conform to these standards, but they must then demonstrate that their devices pass the tests in order to receive a logo for their device. The UPnP Forum provides the means for vendors to certify their devices.

The UPnP Forum also seeks to promote the UPnP standard in the industry and with the general public. It provides a framework for companies to get together and define building block standards: both technical standards, like the UPnP architecture, and legal standards, such as a broadly signed and carefully scoped joint development agreement. These technical and marketing objectives are pursued to advance the entire home networking industry.

A Brief History of UPnP

The core UPnP architecture was originally developed by Microsoft and contributed to the UPnP Forum in the form of the UPnP Device Architecture specification. The specification was approved by UPnP Forum Technical Committee on June 13, 2000. Version 1 of the specification enumerates the UPnP core protocols and establishes the foundation that working committees use to develop their specific devices.

Table 1.1 gives a timeline of activity in the UPnP Forum.

Table 1.1 UPnP Timeline

Date	Event
1/99	UPnP standard publicly announced
10/99	UPnP Forum officially formed
6/00	UPnP version 1 architecture finalized
6/00	Microsoft Windows† ME with UPnP version 1 support ships
7/00	Intel's open source UPnP SDK released
5/01	UPnP version 1 toolkits announced
10/01	Microsoft Windows XP with UPnP version 1 support ships
11/01	First UPnP device standard published
12/01	First UPnP-enabled devices ship
1/02	Microsoft Windows CE with UPnP version 1 support ships

The Committees of the UPnP Forum

The UPnP Forum consists of four organizational elements. Three are permanent committees: the Steering Committee, the Technical Committee, and the Marketing Committee. The fourth is a set of Working Committees formed as needed by participants to define standard device types.

Steering Committee

The UPnP Steering Committee is the high-level directing body of the UPnP Forum. It has about 20 members from various companies, including Microsoft. The composition of the Steering Committee can change over time as new members are added. The Steering Committee provides business leadership and makes decisions for the UPnP Forum. As the organization's management team, the Steering Committee oversees the working committees for defining device descriptions (DCPs). The Steering Committee launched a separate company, the UPnP Implementer's Corporation (UIC), responsible for the certification of devices.

Technical Committee

The UPnP Technical Committee is a group of technical representatives from various companies who process technical issues from working committees. The Technical Committee reviews these issues and produces architectural requirements. They are responsible for the "big picture" technically for the UPnP standard.

Marketing Committee

The UPnP Marketing Committee undertakes joint member promotion of the UPnP standard, including representing the UPnP Forum at industry trade shows.

Working Committees

The nitty-gritty technical work gets done in the Working Committees of the UPnP Forum. These groups define the device descriptions that describe the interfaces that the device provides to the network. The working committees define the syntax and semantics of a particular device type so that implementations of that device type will be interchangeable.

To start a new working committee in the UPnP Forum, members must first make a proposal to the UPnP Steering Committee. The proposal consists of a set of user scenarios to demonstrate the usefulness of the new device type, a schedule of the proposed work, and a commitment from three independent groups to implement the device type. Having multiple independent implementations demonstrates interoperability of the new device type standard. The group is formed with a particular charter, expressed as a set of objectives to be accomplished. Once the group satisfies their charter, its work is complete and the group is disbanded. If the group decides to continue work on a subsequent version of the device type, the group must be re-chartered and meet the same requirements as any other new group to be chartered. This process is summarized in Figure 1.2.

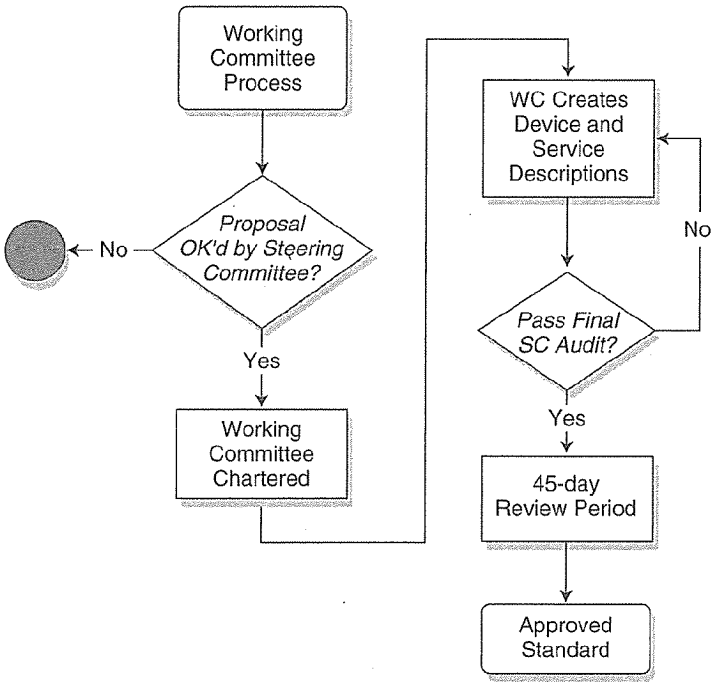


Figure 1.2 Standardization Process Flowchart for UPnP Device Descriptions

All working committees follow the same basic timeline. The working committee members first collaborate to design the device description. This process usually involves weekly conference calls and periodic face-to-face meetings as the group's members work through the issues with the design for the device type. Once the device description is completely designed, test suite development can begin. Sample implementations are typically developed, tracking the development of the standard. Once the test suites are finalized, the implementations can be validated. Working committees often gather for interoperability events to test their independent implementations against one another. After the implementations pass the test suites, the standard moves to the Steering Committee for a final audit. Upon passing this audit, the standard goes into a 45-day review period. At the close of this period, it becomes an approved standard of the UPnP Forum. Figure 1.2 illustrates the standardization process followed by working committees of the UPnP Forum.

The Internet Gateway: The First Certified Device Type

The Internet Gateway Device was the first device standard produced by one of the UPnP Forum working committees. The Internet Gateway Device (IGD) 1.0 working committee included representatives from companies such as Intel, Alcatel, Microsoft, Sony, and Linksys. The IGD supports sharing of Internet connections, advanced connection-management features, management of host-configuration services, and support for transparent Internet access by non-UPnP-certified devices.

Before IGD 1.0, most gateways allowed a single Internet connection to be shared by multiple computers in the home. Unfortunately, this technology prevented many compelling applications, including multiplayer games, file sharing, and real-time communications (such as Internet phone), from working correctly without extra tools and advanced knowledge of gateway and networking configurations.

Currently, the UPnP Forum has many working committees, including Internet Gateways, Audio/Video, Home Automation, Printers and Imaging, Remote I/O, and Security. The committees are formed to meet specific objectives and then are disbanded when their work is done. They are often re-chartered to meet new objectives, but must meet the same criteria as a new working committee, including commitment from

three independent groups to implement the device type. For a list of committees at any give time, visit the UPnP Forum web site at `http://www.upnp.org`.

Security and the UPnP Architecture

UPnP technology helps to make networking automatic—people will bring home networking devices, turn them on, and have them just work, with no technical expertise required. One potential impediment to this vision, however, is the need for security. There is a trade-off between security and ease of use. Implementing security tends to require administration—setting up passwords, defining access control lists, and so on—which gets the user involved again and makes the process of using networked devices less automatic.

In version 1 of the UPnP architecture, there is no built-in security: All UPnP devices on the network can be controlled by any control points. Recently though, a new working committee of the UPnP Forum has been established that is developing a standard security infrastructure compatible with current and future versions of the UPnP architecture.

The UPnP Security Working Committee

The UPnP architecture enables simple networking in the home and small office. "Home and small office" can include many different settings, from single-family homes, apartments, college dorms, and hotel rooms to a local coffee shop providing wireless Internet access for its customers. UPnP devices will enter and leave these dynamic network environments and, as always, unscrupulous people will look for opportunities to take advantage of a lack of security.

The UPnP Security working committee is a new group in the UPnP Forum that has been chartered to provide a security solution for the UPnP architecture that will be common to all device types. The Security working committee includes members from Intel, Microsoft, Siemens, IBM, Sony, and others. In early 2001, the group specified the requirements for a UPnP security solution and defined the user scenarios it intends to support.

Securing the UPnP architecture may eventually expand the use of UPnP technology to new fields, such as providing high-value services. The security solution developed by the working committee will give users choice and control over their network, but will introduce an

inevitable trade-off: security with configuration versus no security with no configuration. The group will undoubtedly try to strike a balance and minimize the configuration required in its security solution.

Requirements of the Security Solution

The UPnP security solution will use standard encryption and digital signature algorithms to protect all of the UPnP protocols. It will include a powerful trust model with non-public key infrastructure authorization certificates, avoiding the heavy infrastructure requirements associated with public key infrastructure (PKI) solutions. It will also be sensitive to the processing capabilities likely to be found on networking devices and will require only moderate processing power to implement.

The UPnP Security working committee will introduce security concepts to the basic UPnP architecture. These additions will likely include principals, permissions, authorization certificates, and access control lists. In addition, the Security working committee will also specify how to secure the basic UPnP protocols, including discovery, control, eventing, and presentation. For example, digital signatures and encryption will be used to maintain confidentiality and to enforce any access control policy.

The SSDP Service Bug

Even with a system that has been designed to be secure, security vulnerabilities can arise from weaknesses in the implementation. These vulnerabilities can result in denial-of-service attacks, preventing systems from being able to offer their services, or provide on opening for an intruder to gain unauthorized access. Microsoft Windows ME and Windows XP contain an implementation of the UPnP protocols and a corresponding API that allows developers to create UPnP control points and devices. Microsoft's Internet Gateway implementation, for example, uses this API to provide the services required of a UPnP Internet Gateway device. Unfortunately, there were two bugs discovered with the implementation of the UPnP protocols shipped with these operating systems[2] (which since have been fixed with subsequent service packs). Both bugs involve how UPnP technology-capable computers handle the discovery of new UPnP devices on the network.

[2] The bugs are also present on Windows 98 and Windows 98SE systems that have the Internet Connection Sharing client installed.

The first bug is an unchecked buffer in the implementation of the Simple Service Discovery Protocol (SSDP). When the SSDP service receives a message from a device that has joined the network, the code processing the messages does not check the input for length. An unchecked buffer, one of the most common and most serious of implementation flaws, allows an attacker to provide more data on an input channel (an SSDP socket in this case) than is expected, overwriting the program stack and allowing the attacker to run any arbitrary code in the context of the application. In this case, the attacker could cause code to be run in the context of the SSDP service, which has system privileges on Windows XP.

The bug's official title was: "Unchecked Buffer in Universal Plug and Play Can Lead to System Compromise" and was documented in Microsoft Security Bulletin MS01-059, which was originally posted on December 20, 2001, at the following URL:

```
http://www.microsoft.com/technet/treeview/
default.asp?url=/|technet/security/bulletin/MS01-059.asp.
```

The second bug introduced by the implementation of the SSDP service provides an opportunity for attackers to use the service to perform two kinds of denial-of-service attacks—a distributed denial-of-service attack where many hosts simultaneously request a device description document from a single host, and a simple denial-of-service attack where many devices may simultaneously request a device description from a single host. The details of these attacks are contained in the Security Bulletin.

The UPnP Implementer's Corporation

The UPnP Implementer's Corporation (UIC) is an independent nonprofit corporation created by the UPnP Steering Committee that administers the UPnP device certification process.

The UIC owns and licenses the UPnP certification mark. Companies with devices that pass conformance tests may license the UPnP logo for use with their device. The UIC licenses conformance tests to UIC members, reviews test results, and issues certificates of conformity to devices that pass the tests. The UIC tests cover the device-dependent features specified in the UPnP device standard and the device-independent features specified in the UPnP version 1.0 architecture.

Summary

- The UPnP standard helps to reduce complexity and simplify home networking for the end user.
- UPnP technology-based products "just work" when they are connected to the network.
- The UPnP architecture is the unifying device abstraction layer for the home of the future, with proxies and bridges spanning to other networks in the home, such as the power line, telephone line, and home entertainment networks.
- With the UPnP architecture, the same kind of open, standard design target we have enjoyed with PC peripherals is coming to the home networking platform.
- UPnP standards will allow devices from different vendors to interoperate.
- UPnP Forum working committees define standard XML-based device and service types that devices may implement.
- Work has begun in the Security working committee of the UPnP Forum to define a security solution for the current and any future versions of the UPnP architecture.

Chapter 2

UPnP Concepts

The mother art is architecture. Without an architecture of our own we have no soul of our own civilization.

—Frank Lloyd Wright

There are a few basic concepts introduced by the UPnP architecture. This chapter introduces these concepts and the underlying UPnP object model, describing each of the different UPnP entities and their corresponding roles and responsibilities. Once you understand this basic object model, you will see some of the common activities that occur on a network of UPnP devices, activities that form the building blocks for futuristic scenarios like those in the previous chapter. The chapter then delves a bit further into UPnP technology, reviewing the UPnP protocol stack and giving a quick overview of each protocol that is part of the UPnP device architecture.

Terminology

Devices, services, and control points are the basic abstractions of the UPnP device architecture. A *UPnP device* can be any entity on the network that implements the protocols required by the UPnP architecture. Because UPnP standardizes the protocols through which a device

communicates rather than the APIs that a programmer uses, any entity that behaves as a UPnP device by speaking the required protocols *is* a UPnP device. Thus, a device either can be a dedicated physical device, such as an Internet gateway, or a logical device, such as a PC, that has implemented the functionality required of an Internet gateway.

A UPnP device contains zero or more services. A *service* is a unit of functionality implemented by a device. Each service has a set of methods, or *actions*, each with a set of optional input and output parameters and an optional return value, much like a function in the C programming language. The specifics of a service, as defined by a UPnP Forum working committee, define each action in detail, listing its required input and output parameters and whether the action returns a value.

An Analogy to Component-based Systems

If you are familiar with a component based system such as Microsoft COM, a UPnP device is like a class object, while the UPnP service is like an interface that the object supports. A UPnP action is similar to a method in a COM interface—it has input and output parameters and may have a return value.

The services that a device must implement are determined by the device's type. The working committees of the UPnP Forum standardize the set of services that particular device types must support.[1] For example, an audio rendering device, such as a CD player, might have a service that provides the ability to play, stop, and pause audio content.

A *control point* is an entity on the network that works with the functionality provided by a device. In the terminology of client/server computing, the control point is the client and the device is the server. Control points can invoke actions on services, providing any required input parameters and receiving any output parameters and possibly a return value. Control points can also request that devices notify them when the device state changes. Figure 2.1 shows a control point invoking an action on a UPnP device. The device has implemented a single UPnP device type that contains two services.

[1] Nonstandard device types may have any set of services and methods as defined by their implementer. It is possible to create proprietary devices and services using UPnP technology, but, by definition, nonstandard services will not be interoperable with devices from other vendors.

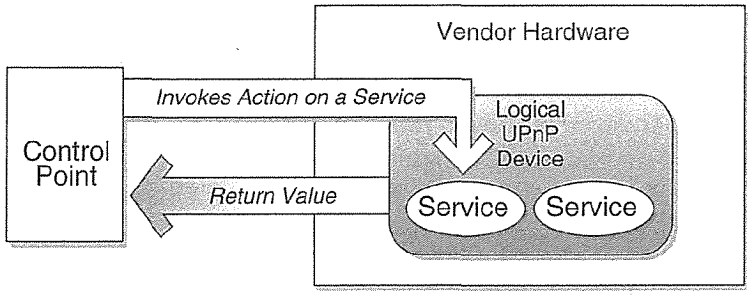


Figure 2.1 Control Point Invoking an Action

Any entity that invokes the services of a UPnP device is a control point. In fact, UPnP technology-enabled devices may have control point functionality built in, so that they can invoke the services or monitor state changes in other devices. With this capability, devices can form a peer-to-peer network where devices take advantage of each other's services.

Figure 2.2 shows a UPnP technology-enabled device that contains a control point that can invoke the services of other UPnP devices.

A device such as an electronic picture frame could implement this pattern by having control point functionality built in, so that when a user presses buttons on the side of the picture frame to navigate through the pictures to be displayed, the control point application on the picture frame retrieves digital images from another UPnP device on the network.

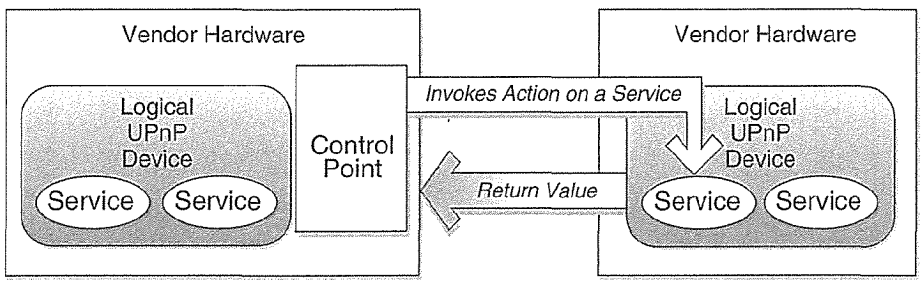


Figure 2.2 Peer-to-Peer: A Control Point in a UPnP Device

Root and Embedded Devices

A UPnP device may also contain other devices. This logical composition of devices gives the UPnP developer flexibility when determining how to implement the device. It also allows the embedded device to be discovered and used independently from the main containing device.

For example, imagine a UPnP television with an embedded UPnP VCR. In this case, the VCR device might have tape transport, tuner, and clock services. The UPnP television device could have a channel control service and a picture adjustment service while containing an embedded VCR device. In UPnP terminology, the top-level device is called the *root device* and the contained device is called an *embedded device,* as shown in Figure 2.3.

A UPnP-enabled device can also implement more than one root device. The UPnP developer has great flexibility in designing the logical structure of devices and services.

State Associated with a Service

Services group the actions provided by UPnP devices. They can also maintain associated states. Like an instance of a C++ class with its member variables, each service may have a *state table*, which is a grouping of its *state variables*. Each state variable has a name, a type, and a value.

UPnP control points can request to receive indications of state variable changes from the service. When a service detects a change to one

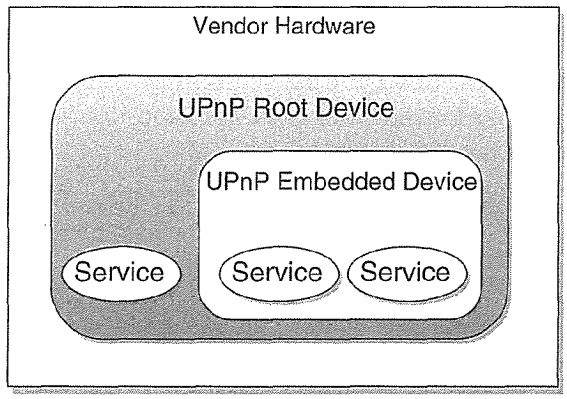


Figure 2.3 Root Device Containing an Embedded Device

of its state variables, the service notifies any registered control points of the change. For example, a service that renders audio tracks might keep a state variable that has the URL of the current track being played. When the track changes, the service sends a state change notification with the new URL to all control points that have registered to receive events from the service.

Figure 2.4 shows how a control point communicates with a service to subscribe to and receive notifications for state variable changes.

UPnP Phases

As you have seen so far, in a network of UPnP devices, control points can discover devices, invoke actions on a device's services, and subscribe to events. Devices, on the other hand, respond to invoked actions and send events when state variables change. To make this basic functionality possible, all UPnP devices follow the same basic pattern, or *phases*, of operation:

- *Addressing.* The device joins the network, acquiring a unique address that others can use to communicate with it.
- *Description.* The device summarizes its services and capabilities in a standard format.

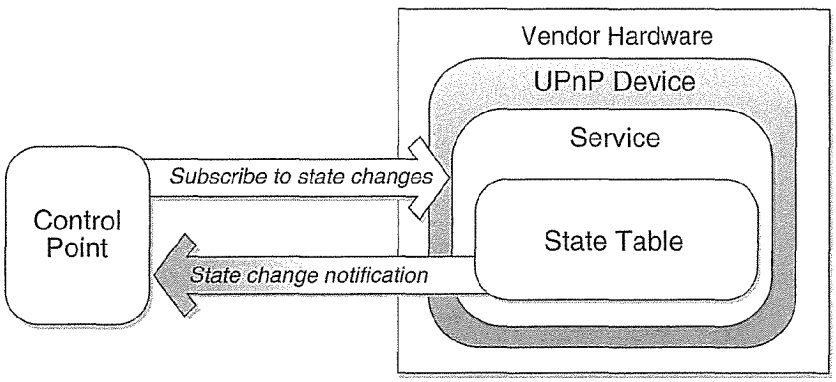


Figure 2.4 State Changes and Notifications

- *Discovery.* The device is found by control points and the device's description information is retrieved.
- *Control.* The device handles requests from control points to invoke actions.
- *Eventing.* The device's services notify registered control points when internal state changes occur.
- *Presentation.* The device optionally provides an HTML-based administrative interface to allow for direct manipulation and monitoring.

Together, these steps define how all UPnP devices behave on a network. Figure 2.5 shows the UPnP phases and their dependencies. A device first acquires an address, then is able to provide a description of its capabilities to control points that have discovered it. Once a control point has discovered a device and retrieved the description of its services, it is able to either control the device, request that it receive notification of events (changes to state variables), or an administrator may manually monitor or configure the device.

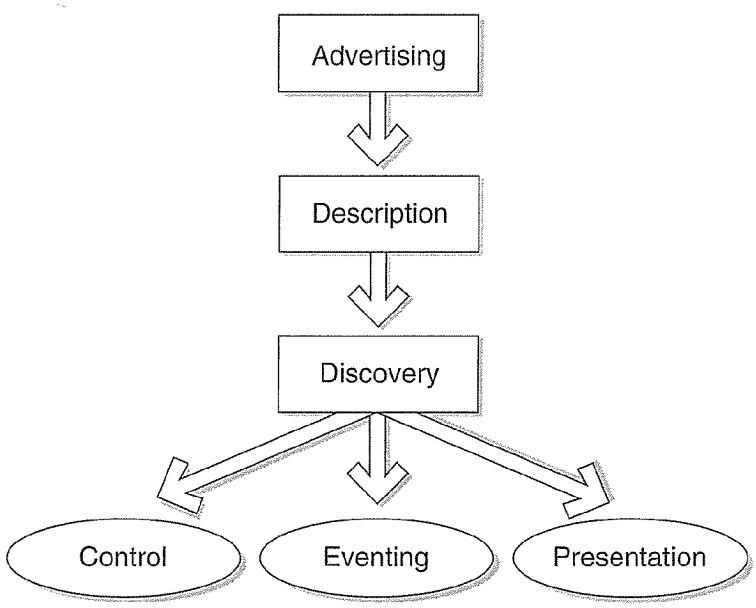


Figure 2.5 UPnP Phases

Each UPnP phase has related network protocols that each device must support. The protocols are described in detail in subsequent chapters, but are briefly introduced here to give you and idea of the scope of a UPnP device.

Addressing

The foundation for UPnP networking is *addressing*, the process by which a device automatically acquires an IP address. As the first step of UPnP, addressing allows devices to join the network and to communicate with other UPnP devices.The addressing protocols built in to UPnP devices allow them to join an IP network dynamically and to acquire an address without user configuration.

Addressing takes into account whether a device is operating in an unmanaged or managed network. An unmanaged, or *ad-hoc*, network is a network where there are no preexisting infrastructure devices (or they are currently inoperable) and the network nodes themselves make up the network. A managed, or infrastructure, network allows devices to acquire an IP address from a DHCP server on the network.

Description

Description allows devices to list the functionality they provide. Descriptions of devices and their services are contained in XML-based *description documents*. The *device description document* contains device information such as manufacturer, make, model, and serial number; a list of services provided by the device; and a list of embedded devices. *A service description* document contains detailed information about the service, the actions it provides, and their parameters and return value.

Discovery

The discovery process enables control points to find devices and services and retrieve information about them. Also, once a device has acquired an IP address, the device may advertise itself and its services on the network. Devices include a URL for their device description document in their advertisements and discovery responses. The URL provides control points with the information they need to retrieve the device and service descriptions, enabling the control points to learn all about the device and the services it offers.

Once a control point has discovered a device and has retrieved the device and service description documents, it may control the device, subscribe to events sourced by the device's services, or retrieve the device's presentation page.

Control

Control is the phase of UPnP where control points invoke the actions provided by a device's services. A device's service receives a control message and acts upon it. The device may change state as a result of the operation, leading to the next UPnP phase, eventing.

Eventing

Eventing allows control points to monitor state changes in devices. The UPnP architecture uses a publisher/subscriber model where control points may subscribe to a service provided by a device. The device's service notifies all registered control points upon changes in state variables. Responding to state changes in this way enables a UPnP network of devices to be a dynamic, responsive, event-driven system.

Presentation

Presentation is the process by which a device presents a browser-based user interface for manual user control and to allow viewing of device status. Each device contains a web server and may provide a web page for browser-based clients. This web page serves as the manual interface for the device as opposed to the device's programmatic control interface. This browser-based interface can be used to control the device, to change operational parameters, to view device and service information, or for any other device-specific functionality implemented by the manufacturer.

The UPnP Object Model

A UPnP developer must be intimately familiar with the basic UPnP objects and their attributes, interfaces, properties, and relationships. Understanding the UPnP object model will give you a conceptual frame of reference for understanding UPnP device implementations.

The Device

By itself, a device does nothing more than provide self-describing information such as the manufacturer, model name, and serial number. A device's set of zero or more services provides its real functionality.

A root device may contain a number of embedded devices. For example, a digital television device might have an embedded audio player device.

Each device can have a set of icons to depict the device in control point user interfaces. The icons are available in different sizes to satisfy different UI requirements.

The device also maintains the URL of its device description document. During the discovery process, the device returns the URL for its description document[2] to allow control points to learn the details of the device.

Figure 2.6 summarizes the relationship between the device, its services and icons, and any embedded devices.

The Service

A UPnP service is conceptually similar to a Java interface, C++ virtual base class, or COM interface in that it provides a set of function signatures that are grouped into a logical whole. In the UPnP architecture, the methods are called actions. In general, each action has a name and may have input and output arguments. Each argument has a name, value, and a direction. The direction may be either input or output (but not both), depending on whether the argument is passed into the action or is returned from the action to the caller. Each action may also have a return value that provides the result of the action.

Services without Actions

Services don't require any actions; they can have only state variables, just as a C++ class can have member variables without having any methods. In this case, the service would only support control points subscribing to state changes and control points querying for the current value of state variables.

[2] The *device control protocol*, or *DCP*, is an earlier name for a device description document. DCP emphasizes the fact that the description document provides the interface, or *protocol*, for the device.

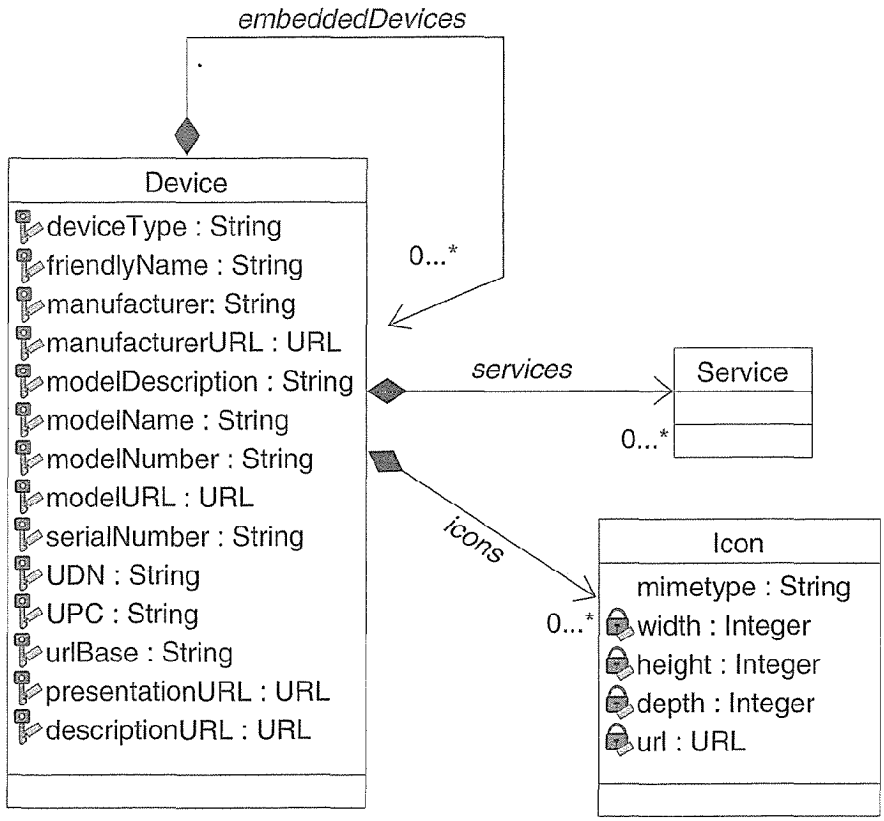


Figure 2.6 Class Diagram: The UPnP Device

The Service's URLs

Each UPnP service has a *service type* Uniform Resource Identifier (URI)[3] that uniquely identifies the service. Standard service types are defined by a UPnP Forum working committee and begin with `urn:schemas-upnp-org:service` followed by a service type suffix, colon, and an integer service version.

For example, the following service type URI is for the UPnP A/V Connection Manager Service, version 1:

```
urn:schemas-upnp-org:service:ConnectionManager:1
```

[3] A Uniform Resource Identifier can be either a name or a URL. Think of it as a unique identifier. Chapter 3 has more information about URIs.

Every service also has a *serviceId* URI that uniquely identifies the service among all of a device's services. No two services may have the same serviceId. For standard services defined by a UPnP Forum working committee, the serviceId must begin with `urn:upnp-org:serviceId:` followed by a serviceId suffix. For example, the serviceId for the Connection Manager Service could be specified as follows:

```
urn:upnp-org:serviceId:cmgr
```

Every service maintains three URLs that provide the information necessary for control points to communicate with services, as follows:

- The `ControlURL` is where control points post requests to control this service. The UPnP vendor specifies one for each device.
- The `EventSubURL` is where control points post requests to subscribe to events. The event subscription URL must be unique within the device; no two services may have the same URL for eventing. If the service has no evented variables, it should not have eventing; if the service does not have eventing, this element must be present but should be empty.
- The `DescriptionURL` tells control points the location from which they can retrieve the service description document. The service description is an XML document that summarizes the information about a service, including each of its actions. This document is returned upon an HTTP GET request.

Each service has zero or more state variables. Each state variable has a name, a type, and a current value. The UPnP Device Architecture specification recommends that a state variable also have a default value, but does not require it. A state variable also has a set of allowed values used to describe the range of permissible values for the variable.

Any of the state variables can trigger events on state changes as determined by the implementer of the service. If the variable does trigger an event when its state changes, it is said to be evented.

Every input argument to an action is associated with one of the service's state variables. This is called the argument's *related state variable*. Figure 2.7 adds the state variable class to the previous diagram of the service and its actions. Only one of an action's arguments may be designated as the action's *return value*. The return value provides the result of the action to the caller.

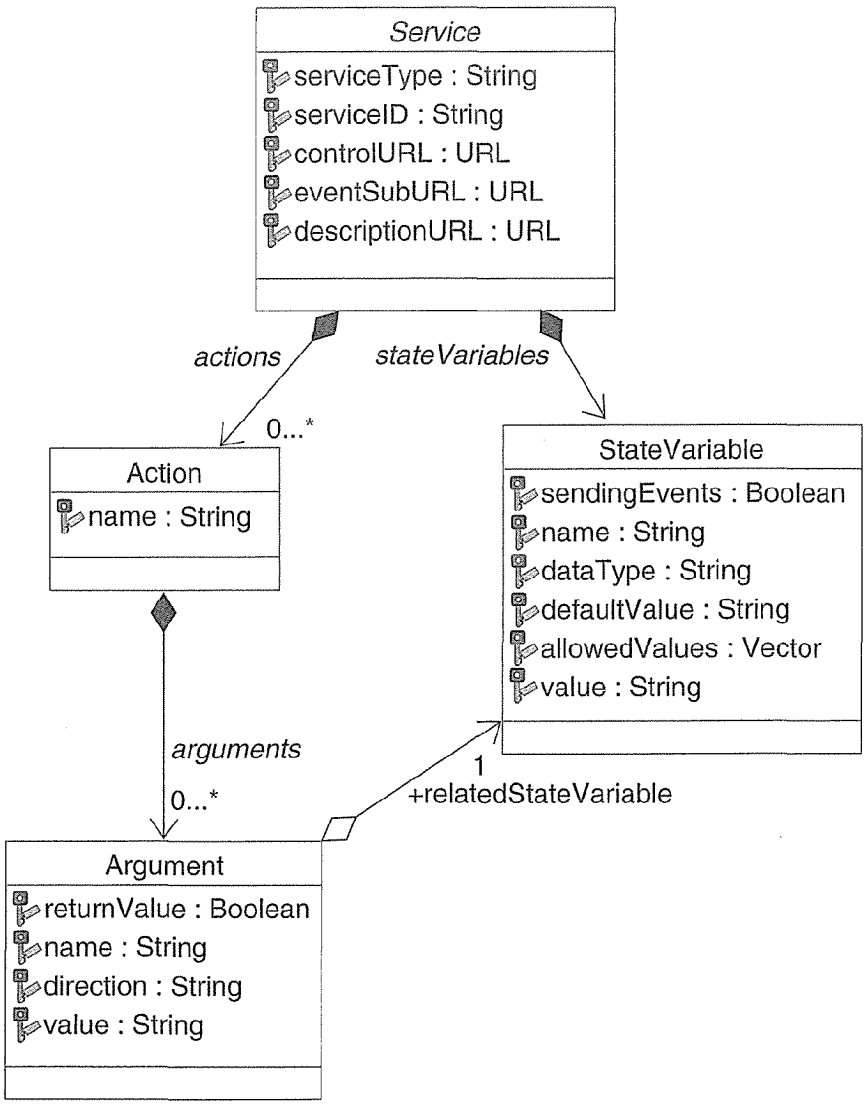


Figure 2.7 Class Diagram: Service, Actions, and State Variables

Events and Subscriptions

UPnP architecture employs an event notification system using a publisher/subscriber model, where control points assume the role of subscribers and services the role of publishers. Services publish event

notifications to control points that have subscribed to receive them. The event notification message from the service has the current value of all of the service's state variables. The UPnP architecture provides no mechanism to subscribe on a variable-by-variable basis.

A subscription is a request from a control point to a service to receive a notification when any of the service's state variables change state. A control point subscribes by sending a subscription request that includes:

- The service of interest
- A URL to which the events can be sent
- A subscription time for the event notification

Accepting the Subscription

If the service accepts the subscription, the service responds with a unique subscription identifier and a duration for the subscription. The unique identifier allows the control point to refer to the subscription in subsequent requests to the service, such as renewing or canceling the subscription. The duration specifies the length of time that the subscription is valid and will be maintained by the service.

Initial Event Message

As soon as possible after the subscription is accepted, the service also sends the first, or initial, event message to the subscriber. This event message contains the names and current values for all evented variables and allows the subscriber to initialize its model of the state of the service.

Modeling a Device

A control point typically maintains an image of the state of devices with which it is working. When the control point receives an event message notifying it of a state change with a device, it updates its internal representation of the device, keeping it in sync with the devices it is monitoring. The control point can use this information in subsequent actions to guide its operation.

State Variables Don't Have to Be Evented

Not all state variables must be evented. A service may designate one or more state variables as *non-evented* and not send event messages to subscribers when these variables change values.

State Change Events

If one or more of a service's state variables are evented, the service publishes updates to the subscribers when the variables change. An event is a message from a service to subscribed control points that is used to keep the control points informed of changes in state associated with the service. Control points subscribe to events and the service notifies them when the event has occurred.

All subscribers are sent all event messages. Subscribers receive event messages for all evented variables, and event messages are sent regardless of the reason that the state variable changed, for instance, in response to an action request or due to an internal state change.

Subscription Expiration

If a subscription expires, the subscription identifier becomes invalid, and the service stops sending event messages to the control point. If the control point tries to send any message other than a subscription request message—a subscription renewal request, for example—the service rejects the message because the subscription identifier is invalid.

Subscription Renewal

All subscriptions must be renewed periodically for the control points to continue to receive notifications (unless they are initially requested for "infinite" duration). To keep the subscription active, a control point must send a renewal message before the subscription expires. The renewal message is sent to the same URL as the original subscription message, but the renewal message does not include a delivery URL for event messages. Instead, the renewal message includes the subscription identifier received in the initial message accepting the subscription.

Subscription Cancellation

When a control point no longer wants to receive events from a service, the control point should cancel its subscription. Generally, canceling a subscription reduces service, control point, and network load. If

removed abruptly from the network, the control point might be unable to send a cancellation message. As a fallback, the subscription eventually expires on its own unless renewed.

The Built-in Web Server

Every UPnP device has a built-in web server that is used as the underlying communication mechanism for much of the device's functionality. The web server receives the messages related to description, control, presentation, and event subscription. It provides access to the device's description document, the service description documents, the device presentation page, device control, and event registration.

UPnP devices build on web technologies to move from a manual web browsing system to a dynamic, distributed system of programmable components on the local network.

The UPnP Architecture and Web Services

The UPnP architecture has a lot in common with Internet-based web services. Devices are like containers for web services, making functionality available in the home that can be composed to create new kinds of distributed applications. In fact, both UPnP and Internet-based web services use the SOAP protocol to invoke methods/actions. However, as you will see in Chapter 5, UPnP uses a local discovery mechanism while web services use an Internet-scale discovery system called UDDI. In the future, as discussed in Chapter 19, UPnP 2.0 will adopt the Web Services Description Language, replacing its current syntax for device and service description. This will bring UPnP another step closer to Internet-based web services.

The UPnP Stack

The TCP/IP protocol suite and HTTP provide the basic network connectivity for the UPnP device architecture. Communication between control points and devices is built upon HTTP as the foundation, over both connection-oriented and connectionless transport protocols. The connection-oriented Transmission Control Protocol (TCP) is used as the transport when a control point is communicating directly with a device or when a service is communicating to a control point. The User Datagram Protocol (UDP) is used when a control point or device needs to communicate to many recipients simultaneously.

Point-to-Point Communication

Each UPnP device provides a set of HTTP URLs for control points to retrieve the device description document, control the device, subscribe to events, and get the devices presentation page. Figure 2.8 shows the basic UPnP protocol stack.

Multicast Communication

A few aspects of the UPnP architecture require the sender to communicate with many recipients simultaneously. For example, UPnP discovery mechanism allows devices to send presence announcements to all of the other devices on the network, letting them know that the device is now available. For this purpose, the UPnP architecture uses a form of HTTP that is sent over multicast UDP, called HTTPMU. In some cases, response messages are sent directly back to the source using HTTP over unicast UDP, called HTTPU. These protocols are illustrated in the Figure 2.9.

Addressing Protocols

The UPnP architecture devices must support both the Dynamic Host Configuration Protocol (DHCP) and the dynamic configuration of private link-local addresses ("Auto-IP"). A device first attempts to contact a DHCP service to acquire an address. If it fails to locate a DHCP server, the device uses Auto-IP, which enables devices to select addresses without a DHCP server being present to assign the address.

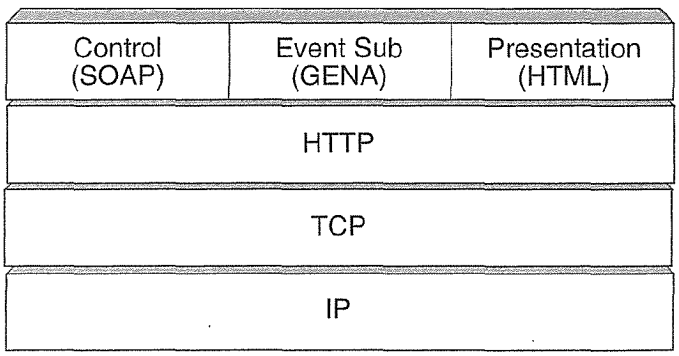


Figure 2.8 The UPnP Architecture Protocol Stack

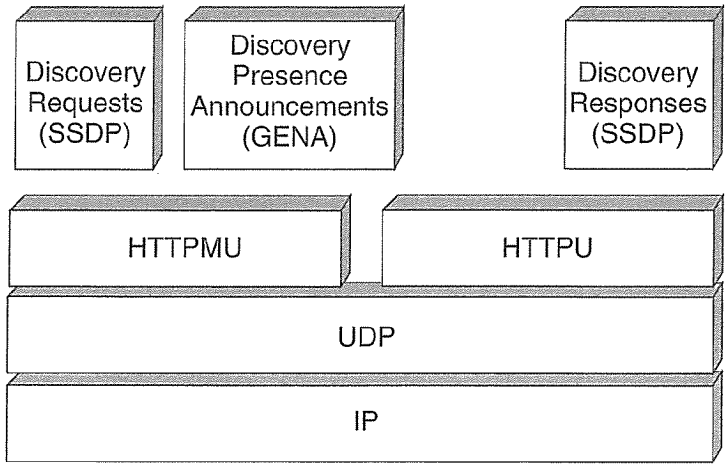


Figure 2.9 HTTPMU and HTTPU Protocol Stack

Summary

- The UPnP architecture is a powerful and flexible software architecture that makes use of existing standard protocols, combining them to provide a zero-administration networking capability for the home or small office network.
- Standard device and service types enable devices from various manufacturers to interoperate.
- All UPnP devices follow the same pattern in their operation and support protocols for addressing, discovery, description, control, eventing, and presentation.
- The UPnP object model that includes devices, services, actions, events, and subscriptions is simple.
- The basic set of operations for this object model allows control points to retrieve information about devices, to invoke device services, and to manage event subscriptions.
- UPnP devices can join an existing managed network or can form ad-hoc networks without the support of networking infrastructure services.

Chapter 6

Description

Calvin:"My life needs a rewind/erase button."

Hobbes: "And a volume control."

—Bill Watterson, *The Authoritative Calvin and Hobbes*

After a UPnP control point discovers a device, it has only the information contained in the discovery message—the device's type, its universally-unique identifier, and a URL to its description document. To find out more about the device, including the services and actions it supports, the control point retrieves description documents from the device. This chapter explains the description process and details about UPnP description documents, with information about

- How devices are described, including vendor-specific information, embedded devices, and URLs for control, eventing, and presentation
- How services are described, including actions, arguments, state variables, and properties of the variables
- How control points retrieve device and service descriptions from devices

UPnP's Description Phase

The description phase is the link between UPnP's discovery and control phases. In the discovery phase, devices advertise their presence to control points on the network, while control points search for devices. Once a control point receives these advertisements and finds a device of interest, it gets the description documents directly from the device to learn more about the device and its services. Once a control point has processed the description documents and understands the device's capabilities, it is ready to control the device, as shown in Figure 6.1.

Description documents allow control points to dynamically adapt to devices. For example, a control point can detect the presence of a preferred vendor-specific service on a device and use that instead of a standard service.

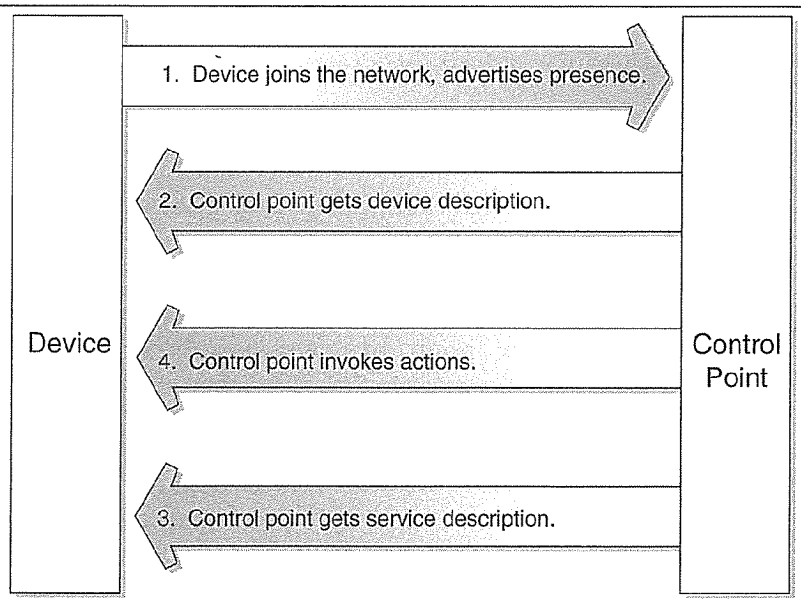


Figure 6.1 Retreiving Device and Service Descriptions

Discovery, Description, and Control

These abstract phases of discovery, description, and control are fundamental to the service-providing process; you must first find a provider, then learn more about the provider to ensure it meets your needs, and then actually receive service from the provider.

For example, have you ever had a problem with your car and needed service? Chances are you looked for an auto mechanic in the local telephone directory. Finding one, perhaps you placed a call to find out more about the company and whether they worked on your type of car. After that, you may have made an appointment to bring the car in and have the work done. If so, you followed the discovery, description, and control phases.

Description Document Standards

Device and service descriptions are simply XML documents that conform to the UPnP Template Language, the XML syntax defined by the UPnP Forum for creating device and service descriptions. This basic template language is used by the various working committees of the UPnP Forum to define standard devices and the services they must contain.

UPnP Forum working committees start with the UPnP Template Language and create description document templates for a particular device type and its services. When implementing UPnP devices, device vendors fill in the placeholders in the description document templates, providing vendor-specific information. Conceptually, the UPnP device template defines the type of device, while the device description document instantiates the template with vendor-provided information.

The device and service templates include information about a standard device and its services, actions, parameters, variables, and so on. For example, the Internet Gateway working committee has standardized the device and service templates for an Internet gateway device. Vendors implementing standards-compliant devices start with these templates defined by working committees and fill in vendor-specific information, perhaps differentiating their devices by included additional services, extending existing services, or embedding additional devices.

The resulting description document returned from a particular UPnP device therefore conforms to a UPnP Forum-defined syntax, implements device and service types defined by a UPnP Forum working committee, and includes information specific to the vendor and hardware, as pictured in the stack in Figure 6.2.

There are a few rules governing UPnP description documents:

- All elements and attributes are case-sensitive.
- All other values, except URLs, are case-insensitive.
- The order of elements is not significant. The elements can be in any order without changing the meaning of the description document.
- Required elements must occur exactly once with no duplicates.
- Recommended or optional elements may occur at most once.
- As specified by the Flexible XML Processing Profile (FXPP), control points must ignore any unknown elements and their subelements or content, and any unknown attributes and their values, when processing device and service descriptions.
- The ampersand character (&) is not allowed in XML. If required, it must be converted into `&` or `%26` (URL escape code).

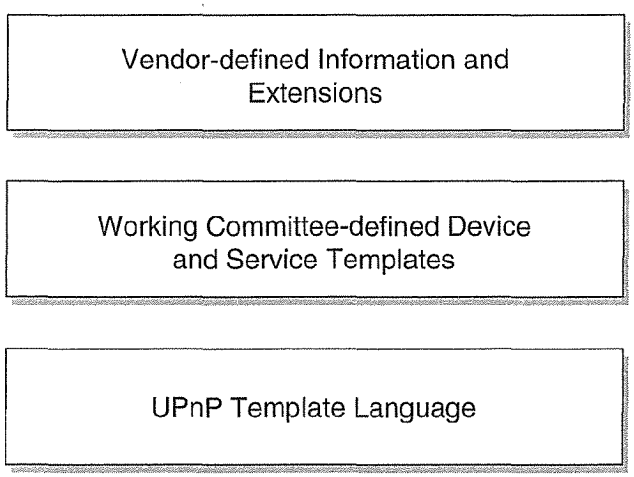


Figure 6.2 Contributions to Description Documents

- Binary data may not be directly included in an XML document. It must first be converted into text using formats such as base64 or binhex. Binary data can be referenced indirectly by using a URL to the data in the document.
- Devices standardized by UPnP Forum working committees must have an integer version number. Later versions must be a superset of earlier versions.

UPnP Device Description Document

A device description document mirrors the logical structure of a device, as discussed in Chapter 2. A device description document has three top-level elements, the <specVersion>, <URLBase>, and <device>, as shown in the following listing (values in *italics* are placeholders for actual values):

Note As we present code and XML through the book, there will be instances where the page width is not wide enough to hold the entire line. Long lines will be cut up into multiple lines, and we'll try to use indentation to make it clear where the line of text is continued.

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

  <specVersion>
    <major>1</major>
    <minor>0</minor>
  </specVersion>
  <URLBase>base URL for all relative URLs</URLBase>
  <device>
    basic device information elements
    service list
    embedded device list
  </device>

</root>
```

As with any XML document, the device document begins with the xml processing directive.

The <root> element of the device description, the top-level representation of the device, has an xmlns tag that includes the standard device

description schema, identified by the URI, `urn:schemas-upnp-org:device-1-0`. This tag references the UPnP Template Language for devices, the schema that defines the syntax for device descriptions.

The <specVersion> element contains two required sub-elements, <major> and <minor> that must be set to 1 and 0, respectively, for UPnP 1.0. The optional <URLBase> element is a single URL that defines the base URL for the device. All relative URLs in the document are appended with this base URL. If the <URLBase> is empty or is not provided, the base URL for the document is the URL from which the device description was retrieved.

Basic Device Information

The device element has many sub-elements that provide details about the device—both manufacturing-related information and information about the functionality provided by the device. This section of the device description document has the following form:

```
<device>
  <deviceType>urn:schemas-upnp-org:device:deviceType:v
    </deviceType>
  <friendlyName>short name for the device</friendlyName>
  <manufacturer>manufacturer name</manufacturer>
  <manufacturerURL>URL to the manufacturer's Web site
    </manufacturerURL>
  <modelDescription>description of the device
    </modelDescription>
  <modelName>model name</modelName>
  <modelNumber>model number</modelNumber>
  <modelURL>URL to model site</modelURL>
  <serialNumber>manufacturer's serial # for the device
    </serialNumber>
  <UDN>uuid:UUID</UDN>
  <UPC>Universal Product Code</UPC>
  <iconList>information about icons</iconList>

  <serviceList>description of services provided by
    the device</serviceList>

  <deviceList>description of embedded services contained
    in this device</deviceList>

  <presentationURL>the URL for the device presentation
    page </presentationURL>

</device>
```

The <device> element contains the sub-elements described in Table 6.1. The strings and URLs in the device description may be, and in some cases should be, localized. In this way, the device can respond with the correct language when a control point specifies a preferred language using the Accept-Language header.

Table 6.1 Device Sub-elements

Element	Required	Type	Description
deviceType	Required	Single URI	The kind of UPnP device this is. Standard device types defined by one of the UPnP Forum's working committees must begin with `urn:schemas-upnp-org:device:` and are followed by the device type suffix, ':', and an integer device version.
friendlyName	Required	String	A short text description of the device. This string should be localized and is specified by UPnP vendor.
manufacturer	Required	String	The name of the manufacturer. This string may be localized and is provided by the UPnP device vendor.
manufacturerURL	Optional	Single URL	The URL for the manufacturer's web site. This URL may be relative to base URL and is specified by UPnP vendor.
modelDescription	Recommended	String	A long text description of the device for an end user. This string should be localized and is specified by UPnP device vendor.
modelName	Required	String	The name of this model of the device. This string may be localized and is specified by UPnP vendor.
modelNumber	Recommended	String	The model number of the device. The string may be localized and is specified by UPnP vendor.
modelURL	Optional	Single URL	The URL of the Web site for this model of the device. The URL may be localized and may be relative to the base URL. Specified by UPnP vendor.

Continues

Table 6.1 Device Sub-elements *(Continued)*

Element	Required	Type	Description
serialNumber	Recommended	String	The serial number of the device. May be localized and is specified by UPnP vendor.
UDN	Required	Single URI	The Unique Device Name for the device. This is a unique identifier for the device that doesn't change, even across device reboots. The UDN must begin with `uuid:` followed by a UUID suffix. The value is specified by the UPnP device vendor.
UPC	Optional	Single UPC	Universal Product Code. A UPC is a 12-digit, all-numeric code that identifies the consumer package. It is specified by UPnP vendor.
presentationURL	Recommended	Single URL	URL for the device presentation page. The URL may be relative to base URL and is specified by UPnP vendor.

A device can have icons associated with it that are used by control point user interfaces to represent the device. A device vendor can supply many icons for a device, with different qualities for different kinds of UIs. After a control point reads a device description document, it can retrieve one of the icons for the device to render in its user interface. To represent this information, the device description has an <iconList> element that consists of a sequence of <icon> elements as shown in the following listing:

```
<device>
…
  <iconList>
    <icon>
      <mimetype>image/format</mimetype>
      <width>horizontal pixels</width>
      <height>vertical pixels</height>
      <depth>color depth</depth>
      <url>URL to icon</url>
    </icon>
   other icons here
</iconList>
…
</device>
```

The <icon> element has the sub-elements listed in Table 6.2.

Table 6.2 Icon Sub-Elements

Element	Required	Type	Description
mimetype	Required	RFC 2387 MIME type	Single MIME image type
width	Required	Integer	Horizontal width of the icon in pixels
height	Required	Integer	Vertical height of the icon in pixels
depth	Required	Integer	Number of color bits per pixel
url	Required	Single URL	Pointer to the icon image. May be relative to the base URL.

The UPnP device architecture recommends one icon in each of the following sizes (width × height × depth): 16 × 16 × 1, 16 × 16 × 8, 32 × 32 × 1, 32 × 32 × 8, 48 × 48 × 1, and 48 × 48 × 8.

The Service List

After the icon list, the device description document contains a list of all of the services the device provides. This is not complete information about the service (that is, what the service description is for), but it does provide the service's type, identifier, URL for retrieving the service description, and a URL for eventing services, as shown in the following listing:

```
<device>
  ...

  <serviceList>
    <service>
      <serviceType>urn:schemas-upnp-org:service:
        serviceType:v</serviceType>
      <serviceId>urn:upnp-org:serviceId:serviceID
        </serviceId>
      <SCPDURL>URL for the service description</SCPDURL>
      <controlURL>URL for controling the service
        </controlURL>
      <eventSubURL>URL for event messages</eventSubURL>
    <service>
    Other standard services here
    Other vendor-defined services here
  </serviceList>

  ...

</device>
```

The required <serviceList> element contains one or more <service> elements with the sub-elements listed in Table 6.3.

Table 6.3 Service Sub-elements

Element	Required	Type	Description
serviceType	Required	Single URI	UPnP service type. For standard service types defined by one of the UPnP Forum working committees, the service type is in the following format: `urn:schemas-upnp-org:service:`*`serviceType:v`*, where *`serviceType`* is the type as defined by the working committee and *v* is an integer service version. Nonstandard service types conform to the following format: `urn:`*`domain-name`*`:service:`*`serviceType:v`*, where *`domain-name`* is an ICANN domain name owned by the device vendor, *`serviceType`* is defined by the vendor, and *v* is an integer service version.
serviceId	Required	Single URI	The serviceId is a identifier for this service that is unique within the device description—no two services may have the same ID. The serviceId for standard services are defined by working committees and have the following format: `urn:upnp-org:serviceId:`*`serviceId`*. Nonstandard services look like: `urn:`*`domain-name`*`:serviceId:`*`serviceId`*, where *`domain-name`* is the ICANN domain name owned by the vendor.
SCPDURL	Required	Single URL	URL for service description (formerly known as Service Control Protocol Definition URL). May be relative to base URL. Specified by UPnP vendor.
controlURL	Required	Single URL	The URL for control messages for the service. This URL may be relative to base URL. Specified in the <URLBase> element. The controlURL is specified by UPnP device vendor.

Continues

Table 6.3 Service Sub-elements *(Continued)*

Element	Required	Type	Description
eventSubURL	Required	Single URL	The URL for event-related messages, such as subscription, renewal, and cancellation. This URL may be relative to base URL. No two services in the device may have the same eventing URL. If the service has no evented state variables, this element must be present, but should be empty. This value is specified by the UPnP device vendor.

The Embedded Device List

A device can contain embedded devices. To describe the embedded devices, the device description document has a `<deviceList>` element that has a `<device>` sub-element for each embedded device. The format of the `<device>` element is the same format as the root device. The `<deviceList>` element is required if and only if the root device has embedded devices.

UPnP Service Description Documents

The service description contains detailed information about a service. Like a device description template, the service description template is created by a working committee of the UPnP Forum. The device vendor fills in the placeholders in the template to create a service description. A service description starts much like a device description:

```
<?xml version="1.0"?>
<scpd xmlns="urn:schemas-upnp-org:service-1-0">
  <specVersion>
    <major>1</major>
    <minor>0</minor>
  </specVersion>
  ....
</scpd>
```

The `<scpd>` element identifies this as a service description (service control protocol document). The `<scpd>` element must have an `xmlns` attribute of `urn:schemas-upnp-org:service-1-0`. This attribute references the UPnP Template Language for services.

Like the device template, the <specVersion> element must have the required <major> and <minor> sub-elements that give the major and minor versions of the UPnP Device Architecture specification to which the device is conforming.

The Action List

After the standard version information, the service description has an <actionList> element that lists of all of the actions (zero or more) that the service supports. In addition to standard actions, UPnP vendors may add their own actions and services. The following listing shows the layout of the <action> elements:

```
<scpd>
  ...
  <actionList>
    <action>
      <name>actionName</name>
      <argumentList>list of arguments</argumentList>
    </action>
    Other standard actions here

     Other vendor-defined actions here

  </actionList>

<scpd>
```

Table 6.4 describes the two <action> sub-elements.

Each action may have zero or more arguments in its <argumentList> element. Each argument in the argument list has a "direction" and may be either an input or output parameter. One of the output arguments

Table 6.4 Action Sub-elements

Element	Required	Type	Description
name	Required	String	Name of action. May not contain any hyphen or '#' characters. For standard actions defined by a UPnP Forum working committee, must not begin with X_ or A_. For nonstandard actions specified by a UPnP vendor and added to a standard service, must begin with X_.
argumentList	May be required		Each action may have zero or more arguments. If an action has arguments, the argumentList element must be present.

may be marked as the action's return value. Each argument must correspond to a state variable (its related state variable). The following listing shows how these elements are described for each argument:

```
<scpd>

  ...
  <actionList>
    <action>
      <name>actionName</name>
      <argumentList>
        <argument>
          <name>formalParameterName</name>
          <direction>in xor out</direction>
          <retval />
           <relatedStateVariable>stateVariableName

                </relatedStateVariable>
        </argument>
       Other arguments
      </argumentList>
    </action>
    Other standard actions here

     Other vendor-defined actions here

  </actionList>

<scpd>
```

Table 6.5 provides the details for the <argument> sub-elements.

The Service State Table

The next major section of service description is the service state table—a list of the service's state variables. State variables are used to model the state of the service at run time. The <serviceStateTable> element of the service description is required and must have one or more state variables. Each state variable is described using the <stateVariable> element. This element has a required sendEvents attribute that may be set to either "yes" or "no" to specify whether the state variable is evented. Each <stateVariable> element has sub-elements to specify the state variable's name, type, and default value, as described in Table 6.6.

There are two ways that the values for a state variable may be specified—by listing the allowed values or by specifying the range of allowed values.

Table 6.5 Argument Sub-elements

Element	Required	Type	Description
name	Required	String	Name of formal parameter. Should be name of a state variable that models an effect the action causes. Must not contain a hyphen character (-, 2D Hex in UTF-8). Should be < 32 characters.
direction	Required		Whether argument is an input or output parameter. Must be in or out, but not both. Any in arguments must be listed before any out arguments.
retval	Optional		Identifies at most one out argument as the return value. If included, must be the first out argument. (Element only; no value.)
relatedStateVariable	Required		Must be the name of a state variable.

Table 6.6 StateVariable Sub-elements

Element	Required	Type	Description
name	Required	String	Name of state variable. Must not contain a hyphen character (-, 2D Hex in UTF-8). For standard variables defined by a UPnP Forum working committee, must not begin with X_ or A_. For nonstandard variables specified by a UPnP vendor and added to a standard service, must begin with X_.
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.
defaultValue	Recommended		Recommended. Expected, initial value. Defined by a UPnP Forum working committee or delegated to UPnP vendor. Must match data type. Must satisfy allowedValueList or allowedValueRange constraints.

The Allowed Value List for String Variables

If a state variable is of type String, the possible values that the variable may take can be specified using a list of allowed values. In this case, the <allowedValueList> element has one or more <allowedValue> sub-elements that specify a legal value for this variable. The following listing shows a state variable with its name, type, and default value, along with a list of allowed values:

```
<scpd>
  ...
  <serviceStateTable>

    <stateVariable sendEvents="yes">
      <name>variableName</name>
      <dataType>variable data type</dataType>
      <defaultValue>default value</defaultValue>

      <allowedValueList>
        <allowedValue>allowed value #1</allowedValue>
        <allowedValue>allowed value #2</allowedValue>
        <allowedValue>allowed value #3</allowedValue>
        Other standard allowed values
      </allowedValueList>

    </stateVariable>
    Other standard state variables
    Other vendor-defined state variables
  </serviceStateTable>
</scpd>
```

The Allowed Value Range for Numeric Variables

For numeric variables, the range of acceptable values can be specified using an <allowedValueRange> element. This element has three sub-elements: <minimum>, <maximum>, and <step>, as follows:

```
<scpd>

  ...
  <serviceStateTable>
    <stateVariable sendEvents="yes">
      <name>variableName</name>
      <dataType>variable data type</dataType>
      <defaultValue>default value</defaultValue>

      <allowedValueRange>
        <minimum>minimum value</minimum>
```

```
            <maximum>maximum value</maximum>
            <step>increment value</step>
          </allowedValueRange>

        </stateVariable>
        Other standard state variables
        Other vendor-defined state variables
      </serviceStateTable>
    </scpd>
```

Retrieving Device and Service Descriptions

Control points retrieve device and service description documents by issuing HTTP GET requests to the appropriate URL. To retrieve a device description, the control point uses the URL contained in the discovery advertisement or response to a discovery query. (Remember, the discovery response has the type, uuid, and URL for the device description document.) The device returns the device or service description in the body of an HTTP response.

For example, consider that a control point has received the URL `http://192.168.1.1/device/toaster` in a discovery response. To get the device description document for this device, the control point issues an HTTP GET as follows:

```
GET device/toaster HTTP/1.1
Host 192.168.1.1:8080
Accept-Language: language preferred by control point
(blank line)
```

The `Host` header gives the name or IP address and optional port of the device description URL. This URL can be from the `Location` header in the discovery message or from the <SCPDURL> element of the device description. If the port value is not supplied, it is assumed to be port 80.

The `Accept-Language` header specifies the preferred language of the control point. If the device does not have a description available in this language, it may return the device description in a default language. The values for this header are from the set of RFC 1766 language tags.

This HTTP request to the device's URL causes the device to return its device description in the following HTTP response (values in *italics* are placeholders for actual values):

```
HTTP/1.1 200 OK
Content-Language: language used in description
Content-Length: length of body in bytes
```

```
Content-Type: text/xml
Date: when responded

<?xml version="1.0"?>
XML device description
```

Like other UPnP responses, the device must respond within 30 seconds, including expected transmission time. Once the control point has the device description, it can parse the document and have URLs to retrieve each of the service descriptions. A similar request is made to each of the SCPDURLs.

```
GET service/foobar HTTP/1.1
Host: 192.168.1.1:8080
Accept-Language: language preferred by control point
(blank line)
```

This HTTP request to the SCPDURL results in the following:

```
HTTP/1.1 200 OK
Content-Language: language used in description
Content-Length: length of body in bytes
Content-Type: text/xml
Date: when responded

<?xml version="1.0"?>
XML service description
```

The body of this response is a UPnP device or service description as explained in detail above. Table 6.7 lists the headers used in a description document response.

Table 6.7 Description Document Response Headers

Header	Required	Type	Description
Content-Language	Required if and only if an Accept-Language header was provided in the request.	RFC1766 language tag	Description documents can be returned in one of many languages.
Content-Length	Required	Integer	Length of body in bytes
Content-Type	Required	Must be `text/xml`	Description documents are XML documents
Date	Recommended	RFC1123 date	Date the response was generated.

Validity of the Information in Description Documents

Device and service descriptions are valid for as long as the device remains on the network with a discovery advertisement that has not expired. As long as the discovery advertisements from a device have not expired, a control point may assume that the device and its services are available. If the discovery advertisement expires or is canceled (receives a "bye-bye" message) and a new one is issued, the control point must reread the description documents for the most current information about the device.

Rules on using the information are:

- If a device cancels its advertisements and re-advertises, it may change information in its description documents.
- Control points must not assume that the descriptions remain unchanged.

Summary

- The description phase of UPnP comes after discovery and enables control points to find out details about devices and the services that they implement. Once control points have this information, they are ready to invoke actions on the device's services.
- Description enables subsequent phases of UPnP: control, eventing, and presentation.
- Device and service description documents are XML documents that follow a standard schema defined by the UPnP Forum.
- The contents (elements) of specific device and service descriptions are defined by various working committees of the UPnP Forum and correspond to the required information that a standard UPnP device must have.
- Description documents are retrieved using a simple HTTP GET to a device's URL or a service's SCPDURL.

Chapter 7

Control

There's nothing remarkable about it. All one has to do is hit the right keys at the right time and the instrument plays itself.

—Johann Sebastian Bach

After a UPnP device has acquired an IP address and advertised its presence on the network, control points can then discover the device and invoke any of the actions provided by the device's services. In UPnP terminology, this invocation process is called *control*.

The control protocol used between UPnP control points and devices is the Simple Object Access Protocol (SOAP). As a UPnP developer, you will not typically be required to compose SOAP messages directly (this will be handled by the particular SDK you are using), but understanding SOAP and its operation will be helpful to you when debugging and tracing control messages between control points and devices.

This chapter begins by discussing the important characteristics of remote procedure call (RPC) mechanisms such as SOAP. Then you will take a closer look at SOAP and how it brings remote procedure calls into the realm of web-based resources. The chapter presents the details of the protocol and will show you some sample messages. The last part of the chapter discusses how SOAP is used by UPnP devices and discusses some potential pitfalls to be aware of.

Validity of the Information in Description Documents

Device and service descriptions are valid for as long as the device remains on the network with a discovery advertisement that has not expired. As long as the discovery advertisements from a device have not expired, a control point may assume that the device and its services are available. If the discovery advertisement expires or is canceled (receives a "bye-bye" message) and a new one is issued, the control point must reread the description documents for the most current information about the device.

Rules on using the information are:

- If a device cancels its advertisements and re-advertises, it may change information in its description documents.
- Control points must not assume that the descriptions remain unchanged.

Summary

- The description phase of UPnP comes after discovery and enables control points to find out details about devices and the services that they implement. Once control points have this information, they are ready to invoke actions on the device's services.
- Description enables subsequent phases of UPnP: control, eventing, and presentation.
- Device and service description documents are XML documents that follow a standard schema defined by the UPnP Forum.
- The contents (elements) of specific device and service descriptions are defined by various working committees of the UPnP Forum and correspond to the required information that a standard UPnP device must have.
- Description documents are retrieved using a simple HTTP GET to a device's URL or a service's SCPDURL.

Remote Procedure Calls

In a computer program running as a single process on a single computer, individual program components can refer to each other directly because they are in the same memory address space. If a component needs to access a data structure, the component simply references it directly. Also, when calling other functions, parameters are simply pushed on the stack or passed in registers, and control is passed directly to the function. However, when programs are partitioned into many components on potentially many different machines, the components can no longer directly reference each other, but must use some intermediate inter-process message-passing system to achieve the same result. Like a function or procedure call in a nondistributed program, the client process issues a request to a server and the response is transmitted back to the client from the server. Each component becomes like a mini-server, exposing its programmatic interface and allowing clients to invoke functions from across the network.

Remote procedure call mechanisms that support this distributed computing model must, at a minimum, have the following:

- *A Message-Based Protocol.* Includes requests, responses, and error responses to allow a caller to remotely invoke a function, procedure, or method at the server.
- *Platform-Independent Data Representation.* Used to package function call parameters. The representation may include predefined types to allow the user to specify commonly used types, such as an integer or a character sequence, and may have extensibility mechanisms to allow user-defined types.

The model may also have optional features such as:

- *Security.* Using various means such as digital signatures, signed messages, encryption, and client authentication can achieve confidentiality, integrity, and authentication.

RPC implementations can range from a simple, low-budget request/response protocol with fixed data types and no security features to a full-blown system with user-definable types and secure messaging using digital signatures and a public key infrastructure. In either case, the basic functionality is the same: Make a simple call to a network-based remote entity as if it was a local component, as shown in Figure 7.1.

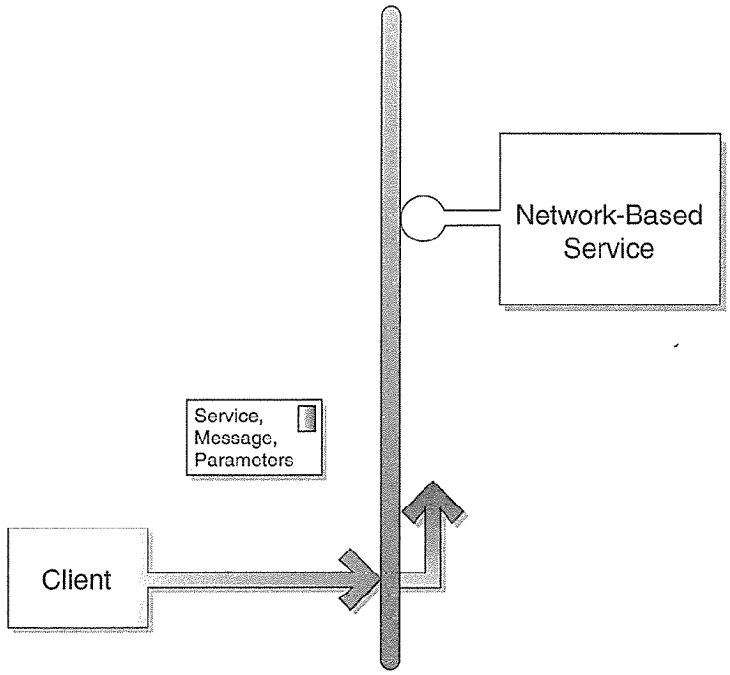


Figure 7.1 Client Issuing an RPC to a Network-based Service

Remote procedure call mechanisms have been around for a long time and the issues surrounding them are well understood. However, bringing these ideas to the web, including the home and small office, is a recent innovation.

The Simple Object Access Protocol

The Simple Object Access Protocol (SOAP) is a protocol that brings together XML and HTTP to provide a Web-based messaging and remote procedure call mechanism. XML is used to express the contents of the messages, while HTTP is used to send the messages to their destination. This section describes SOAP in general and not specifically how SOAP is used to allow control points to invoke the services provided by UPnP devices. Later sections explain the conventions introduced by the UPnP architecture for using SOAP as its control protocol.

SOAP is specified as a set of conventions that govern the format and processing rules of SOAP messages. SOAP consists of four parts:

- *The SOAP envelope.* An XML schema that defines a framework for describing what is in a message, how to process it, and whether it is optional or mandatory.
- *The SOAP encoding rules.* Another XML schema that defines a set of rules for expressing instances of application-defined data types.
- *The SOAP binding.* A convention for using different transport protocols. SOAP can potentially be used in combination with a variety of other transport protocols. (However, SOAP is most commonly carried by HTTP.)
- *The SOAP RPC representation.* A convention for representing remote procedure calls and responses.

The SOAP message is the basic unit of communication between peers. SOAP messages are written in XML, making SOAP platform-independent (any system capable of creating and parsing XML documents can send and receive SOAP messages). Because of the power of XML, SOAP messages can be fairly complex in structure and can transmit highly complex data types.

SOAP Namespaces

XML namespaces, like namespace constructs in programming languages like C++ and Java, ensure uniqueness among XML elements, avoiding collisions among elements from different sources with the same name. There are four namespaces used in SOAP, each of which is independent of the others, as shown in Table 7.1.

Table 7.1 SOAP Namespaces

Namespace	Description
http://www.w3.org/2001/06/soap-envelope	SOAP envelope namespace. The envelope is the outermost container for SOAP messages.
http://www.w3.org/2001/06/soap-encoding	SOAP encoding namespace. The encoding rules specify how to encode data types in SOAP messages.

Continues

Table 7.1 SOAP Namespaces *(Continued)*

Namespace	Description
http://www.w3.org/2001/XMLSchema-datatypes	XML schema for data types. The basic set of types for SOAP messages.
http://www.w3.org/2001/XMLSchema-instance	XML schema for instances defines several attributes for use in any XML documents.

The SOAP Message Envelope

The *SOAP message envelope* is the outermost container of SOAP messages. It is a well-formed XML document that comes right after the standard HTTP headers in the body of an HTTP message. The <Envelope> element contains two child elements: <Header> and <Body>. The SOAP <Header> element is optional, but the <Body> element is required. The <Header> and <Body> elements each contain SOAP blocks, which are valid XML data. A block within the SOAP header is called a header block and a block within a SOAP body is called a body block.

The SOAP Header Element

The *SOAP header* is an optional element for carrying auxiliary information for authentication, transactions, and payments. It is a collection of zero or more SOAP blocks. The SOAP <Header> element can contain an unlimited number of child elements that support the message in some way and enable developers to extend SOAP messages in very powerful ways.

The SOAP Body Element

The *SOAP body* is a collection of zero or more SOAP blocks. The SOAP <Body> element contains the core of the message—a remote method call and its associated arguments, a method response, or error information for failed calls.

By convention, a method response is contained in a child element that is named by appending the word "Response" to the name of the remote method.

Figure 7.2 illustrates the composition of a SOAP message.

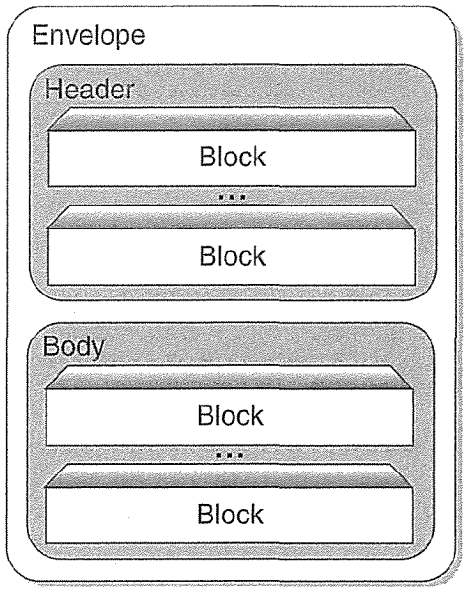


Figure 7.2 SOAP Encapsulation Model: Envelope, Header, Body, and Blocks

For example, the following SOAP message has a body that has a call to a `GetInformation` method of some fictitious Web service. This particular message does not use a header.

```
<?xml version="1.0" encoding="utf-8" ?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/
soap/envelope/" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/
XMLSchema">
  <soap:Body>
    <MethodName />
  </soap:Body>
</soap:Envelope>
```

SOAP Encoding Rules

The SOAP encoding rules allow you to actually pass useful data to a SOAP endpoint by defining how to encode data types in a SOAP message. The encoding rules provide a representation for a type system that can be used to encode data into XML. The encoding rules attempt to include many common features found in programming language type

systems, databases, and semi-structured data, including simple scalar types and compound types.

You don't *have* to use the SOAP encoding rules for encoding your data, but they do provide a complete type system for you. This chapter does not go into detail on the types available in the SOAP encoding rules. Table 7.2 lists some of the simple scalar types available.

Table 7.2 SOAP Scalar Types

Type	Example
32-bit signed integer	-27
Boolean	0 or 1
ASCII string	this is a string
Signed floating point number	-27.327
date/time	2001-03-27T00:00:01-08:00
base64-encoded binary	eW91IGNhbid0IHJlYWQgdGhpcyE=

Conventions for SOAP over HTTP

SOAP is not dependent upon any particular underlying transport protocol and can be carried by a variety of transport protocols, including HTTP, SMTP, and FTP. However, for each particular transport protocol, conventions must be established to allow the transport to carry SOAP's XML payload. For example, binding SOAP to HTTP is relatively simple because SOAP's request/response message model matches HTTP's model, making it easy to encapsulate SOAP within HTTP. However, conventions such as how to set HTTP's content type, what additional HTTP headers should be used, and how errors are handled also need to be agreed upon.

The SOAP HTTP Request

A SOAP message is sent as an HTTP `POST` with the content type set to `text/xml` to indicate that the content of the message is an XML document. SOAP request messages include a new HTTP header, `SOAPAction`, whose value is a URI that indicates the intention of the SOAP request—to let the receiver know that this is a message for a particular service, for example. SOAP does not require this field to point to an actual web

resource, however. The information is provided to allow intermediate processors, such as a web server receiving all SOAP messages, to route or filter incoming SOAP messages.

The SOAPAction header can be set to either an empty string, the name of the SOAP method, or it can have no value. The empty string ("") means that the intent of the SOAP message is provided by the HTTP Request-URI, and no value means that there is no indication provided for the intent of the message.

The following example shows a sample SOAP request. The example highlights the HTTP POST request, the Content-Type header set to text/xml, and the SOAPAction header set to indicate you're calling a fictitious book checkout service. (As you will see shortly, the UPnP device architecture defines a particular format for the value of the SOAPAction header, specifying the service and action to invoke.) The body of the message depends on the particular action being invoked.

```
POST /Checkout HTTP/1.1
Host: library.intoast.com
Content-Type: text/xml; charset="utf-8"
Content-Length: length of body in bytes
SOAPAction: http:// library.intoast.com/checkout

<s:Envelope
xmlns:s="http://www.w3.org/2001/06/soap-envelope/
    " s:encodingStyle="http://www.w3.org/2001/06/
    soap-encoding/">
  <s:Body>
  Body of message here...
  </s:Body>
</s:Envelope>
```

M-POST, HTTP Extensions, and the MAN Header

SOAP uses the HTTP Extension Framework to extend HTTP. To ensure that the introduced SOAPAction header is not confused with other HTTP extensions, SOAP conforms to the HTTP Extension Framework by specifying a unique URI in the MAN header and attaching the prefix M- to the POST method. Using the M-POST method requires the HTTP server to find and understand the URI in the MAN header and to understand the SOAPAction header.

The SOAP specification requires that requests must first be attempted *without* the MAN header or M- prefix. If the request fails with 405 Method Not Allowed, then a second request must be sent using the MAN header and M- prefix. If that request fails with 501 Not Implemented or 510 Not Extended, then the request fails.

The SOAP HTTP Response

The SOAP HTTP response, like the request, is an XML document contained in a standard HTTP message whose content type is `text/xml`. The XML document for the response is structured just like the request, using the <Envelope> and <Body> tags, but the Body itself contains the encoded method result instead of the method call. Methods that do not return a value (void methods) simply omit the <return> part of the Body.

SOAP over HTTP follows the semantics of the HTTP status codes. For example, 2xx status code[1] indicates success—the client's request was successfully received, understood, and accepted. There are five values for the first digit of the HTTP return code, as summarized in Table 7.3.

The following example shows a successful SOAP response, including the 200 result code, the `Content-Type` header, and the SOAP envelope containing the response.

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset="utf-8"
Content-Length: length of body in bytes

<s:Envelope xmlns:s='http://www.w3.org/2001/06/
    soap-envelope'>
  <s:Body>
  Body of response message here...
  </s:Body>
</s:Envelope>
```

Table 7.3 HTTP Return Code Categories

Number Range	Meaning	Description
2xx	Success	The action was successfully received, understood, and accepted.
3xx	Redirection	Further action must be taken in order to complete the request.
4xx	Client Error	The syntax of the request is invalid or cannot be fulfilled.
5xx	Server Error	The server failed to fulfill an apparently valid request.

[1] The first digit of the status code defines the kind of response. The last two digits are not significant in categorizing the response, but provide further information about the error.

SOAP Exceptions

If an error occurs while the server is processing the SOAP request, the server issues a `500 Internal Server Error` response and returns a SOAP message containing a SOAP fault in the response. The SOAP fault appears within the Body of a SOAP message and carries error and/or status information. There are four sub-elements of the `<Fault>` element, `<faultcode>`, `<faultstring>`, `<faultactor>`, and `<detail>`.

The `<faultcode>` sub-element provides a value that indicates the reason for the fault and is intended to provide information that an application can use to recover from the fault without user intervention. The `<faultcode>` values include those listed in Table 7.4.

The `<faultstring>` sub-element contains an explanation of the fault. Typically, the `<fault>` sub-element would be used by the application to recover from the fault, while the `<faultstring>` sub-element contains more detailed information for the user to assist in recovering from the error.

The `<faultactor>` sub-element is a URI that identifies the source of the fault.

The `<detail>` sub-element carries application-specific error information related to the SOAP Body.

The listing in Figure 7.3 shows all of the SOAP Fault sub-elements in a SOAP fault returned to a caller.

Table 7.4 Values of the `<faultcode>` Sub-element

Name	Description
VersionMismatch	Returned when the server detects an invalid namespace in the request.
MustUnderstand	Returned when the server does not understand an immediate child element of the SOAP header and the SOAP mustUnderstand attribute was set to "1" on the request.
Client	Returned when the message was not correctly formed or did not contain required information.
Server	Returned when the server cannot complete the request for some reason. The message itself is not the problem, but the server's processing of it.

```
<s:Envelope

xmlns:s='http://www.w3.org/2001/06/soap-envelope'
xmlns:f='http://www.w3.org/2001/06/soap-faults'>
  <s:Body>
    <s:Fault>
      <faultcode>MustUnderstand</faultcode>
      <faultstring>One or more mandatory headers not
        understood</faultstring>
      <faultactor>fault actor</faultactor>
      <detail>more detail </detail>
    </s:Fault>
  </s:Body>
```

Figure 7.3 A Sample SOAP Fault

The Control URL

In a UPnP device description document, each service element has a <controlURL>—an element that contains a URL where all control messages for that service are to be sent. Control points send SOAP-based control messages to this control URL and, in response, the service returns any results or errors from the action. The following listing highlights the location of the <controlURL> element in a device description document.

```
<xml version="1.0"?>
<root xmlns="urn:schemas-upnp-org:device-1.0">
  <specVersion>
    <major>1</major>
    <minor>0</minor>
  </specVersion>
<URLBase>base URL for all of the relative URLs</URLBase>
device>
  other device description elements
  <serviceList>
    <service>

      <controlURL>URL for control requests</controlURL>

    </service>
  </serviceList>
</root>
```

Action Request

To invoke an action using the POST method, a control point must send a request in the following format to the service's controlURL.

```
POST controlURL HTTP/1.1
Host: controlURL host:port
Content-Length: length of body in bytes
Content-Type: text/xml; charset="utf-8"
SOAPAction: "urn:schemas-upnp-org:service:
  serviceType:v#actionName"

<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>
```

The request line for this message uses the POST method. The controlURL sub-element will be the path component of the URL for control for this service. Table 7.5 lists the headers used when invoking an action on a service.

The body of the HTTP message consists of a SOAP envelope. The required namespace attribute for this element,

```
http://schemas.xmlsoap.org/soap/envelope/
```

includes the schema for the SOAP envelope. The encodingStyle attribute must also be present and must be

```
http://schemas.xmlsoap.org/soap/encoding/
```

All SOAP requests follow this pattern.

The Body element contained in the SOAP envelope contains the body of the action request and is qualified with the SOAP envelope namespace. It contains the required <actionName> sub-element, which contains the name of the action in the service the caller wishes to invoke. This element must include the XML namespace of the service being called. The format for this attribute is urn:schemas-upnp-org:service:serviceType:v.

Table 7.5 Action Request Headers

Header	Required	Type	Description
Content-Length	Required	Integer	Length of the body of the message in bytes.
Content-Type	Required	Must be `text/xml`	Should also include the `charset` attribute to specify the character encoding used, such as UTF-8.
Host	Required	Domain name or IP address and optional port of the control URL for the service	Given in the `<controlURL>` sub-element of the service element in the device description. If the port is not supplied, port 80 is assumed.
Man	Required for M-POST only	Value is set to the XML schema for SOAP envelopes. Contains a namespace directive that is then used on the `SOAPAction` header.	No Man header is required with the `POST` method. Required with `M-POST`.
SOAPAction	Required	Single URI.	Must be the service type, "#", and name of action to be invoked, enclosed in double quotes. If used in a request with method `M-POST`, the `SOAPAction` header name must be qualified with HTTP namespace defined in the `Man` header.

If the action has arguments, each argument is provided with the name of the argument in enclosing tags and the value of the argument within the tags. The data types of the arguments are defined by the UPnP service description.

As discussed in the previous section on SOAP, if a request with a `POST` method is rejected with a `405 Method Not Allowed` message, then the

control point must send a second request with the M-POST method and a MAN header, as follows:

```
M-POST controlURL HTTP/1.1
Host: controlURL host:port
Content-Length: length of body in bytes
Content-Type: text/xml; charset="utf-8"
Man: "http://schemas.xmlsoap.org/soap/envelope/"; ns=s
s-SOAPAction: "urn:schemas-upnp-org:service:
  serviceType:v#actionName"

message body same as for POST
```

The request line for this message uses the M- prefix as defined by the HTTP Extension Framework. The headers are the same as the POST method, except that there is now a MAN header that must have the value

```
http://schemas.xmlsoap.org/soap/envelope/
```

The ns directive in the MAN header value defines the namespace for other SOAP headers (SOAPAction in this case). The SOAPAction header (qualified with the namespace defined in the MAN header) must be the service type and version, "#", and the name of the action to invoke, all enclosed in quotes.

The message body for a request using the M-POST method is the same as that for the POST method.

Action Response

A service has 30 seconds to complete the action and respond to the control point, including expected transmission time. According to the UPnP device architecture, actions that are expected to take longer than this should return early and send an event when the action completes. A service responds using the following format:

```
HTTP/1.1 200 OK
Content-Length: length of body in bytes
Content-Type: text/xml; charset="utf-8"
Date: when response was generated
Ext:
Server: OS/version UPnP/1.0 product/version

<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>output argument value</argumentName>
      other output arguments and values, if any
    </u:actionNameResponse>

  </s:Body>
</s:Envelope>
```

The initial line for the response message is the typical HTTP success response. Headers used for an action response are listed in Table 7.6.

The action response is contained in a typical SOAP <Body> element within a SOAP envelope. By convention, the response to a particular action is named by appending Response to the action name. The action response contains any output argument values returned from the action in the same format described previously for input. The first output argument is defined to be the actions return value.

Table 7.6 Action Response Headers

Header	Required	Type	Description
Content-Length	Required	Integer	Length of the body of the message in bytes.
Content-Type	Required	Must be `text/xml`	Should also include the character encoding used (e.g., UTF-8).
Date	Recommended	RFC1123 date	When the response was generated.
Ext	Required	no value	Confirms that the `MAN` header was understood.
Server	Required	String	Concatentation of OS Name, OS version, UpnP/1.0, product name, and product version.

Action Error Response

If the service is not able to successfully complete the action, it sends an error return message to the control point. Error responses are sent in the following format:

```
HTTP/1.1 500 Internal Server Error
Content-Length: length of body in bytes
Content-Type: text/xml; charset="utf-8"
Date: when response was generated
Ext:
Server: OS/version UPnP/1.0 product/version

<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>
```

The initial response line provides the 500 error code to indicate an unsuccessful request. The headers for an error response are the same as those for a successful response: `Content-Length`, `Content-Type`, and so on.

The body of the message is a SOAP response with the typical `<Envelope>` and `<Body>` elements.

The differences start with the highlighted SOAP `<Fault>` element. An error response for a failed UPnP action includes the required elements of a SOAP fault: `<faultcode>`, `<faultstring>`, and `<detail>`. The `<faultcode>` element must have a value of `Client` qualified with the SOAP namespace. The `<faultstring>` element must be `UPnPError`. The `<detail>` element contains a `<UPnPError>` sub-element that itself has two sub-elements, `<errorCode>` and `<errorDescription>`. Table 7.7 summarizes the possible values for the `<errorCode>` and `<errorDescription>` elements.

Table 7.7 <errorCode> and <errorDescription> Values for a Failed Action

errorCode	errorDescription	Description
401	Invalid Action	The service has no action of the name provided.
402	Invalid Args	Problem with the input arguments: not enough arguments, too many arguments, wrong name, or wrong type.
403	Out of Sync	Out of synchronization.
501	Action Failed	Current state of service prevents invoking the action.
600–699	UPnP Forum defined	Common action errors. Defined by UPnP Forum Technical Committee.
700–799	Depends on device type	Action-specific errors for standard actions. Defined by UPnP Forum working committees.
800–899	Vendor-defined	Action-specific errors for nonstandard actions. Available to be defined by the UPnP device vendor.

QueryStateVariable

In addition to invoking actions on a device's service, control points may also directly query the service for the value of a state variable. To do this, a control point can use the `QueryStateVariable` action. All services support this action implicitly. A control point can use `QueryStateVariable` to get the value of a single state variable. However, use of this method is discouraged in the Universal Plug and Play Vendor's Implementation Guide (<http://www.upnp.org/download/UPnP_Vendor_Implementation_Guide_Jan2001.htm>), a document that contains clarifications to the UPnP Device Architecture specification.

The Implementation Guide recommends that implementers should only invoke actions explicitly defined for the particular service type and should reserve using `QueryStateVariable` for limited testing scenarios. Using explicitly defined functions instead of `QueryStateVariable` has the following benefits:

- *Improved efficiency.* All of a service's state variables can be queried using a single action.
- *Clearer definition of intended use of a service's state variables.* The read and write access patterns are explicit in the service's actions.

- *Improved interoperability.* Control points will use the service only through its standard interface and not in ways unintended by the service designer.
- *Reduced implementation size.* Services do not have to maintain memory for certain types of non-evented state variables.

QueryStateVariable may be removed from future versions of the UPnP standard. Use it with caution.

Summary

This chapter has covered a lot of ground, from RPC basics, to an overview of SOAP, to details about how UPnP control points use SOAP to invoke actions on a device's services. The most important points are:

- SOAP is a messaging and remote procedure call technology that can be used over a variety of transports, but is primarily used over HTTP.
- Along with its use of XML, SOAP is a good choice for a web-based RPC mechanism.
- UPnP control points use SOAP to invoke actions provided by services contained on a device.
- UPnP introduces a few more conventions when using SOAP, such as the `controlURL`, `QueryStateVariable`, and the contents of the `POST` method and `SOAPAction` header.

Chapter 17

UPnP Audio/Video

This chapter introduces one of the most popular areas of UPnP device development: UPnP Audio/Video, or UPnP A/V for short. The UPnP A/V working committee has defined an architecture for distributing digital audio and video using UPnP. You'll learn the specific device types supported in UPnP A/V, what services and actions those devices are required to support, and how they work together to support UPnP technology-based media distribution in the home.

Problem Statement

In today's home, a wide variety of disparate devices exist that support the storage, playback, and rendering of content. DVD players, VCRs, and televisions are a staple of the consumer home. Yet these devices are typically tied to a specific format type and don't present the same network-based discovery, configuration, and control that UPnP devices can offer. The goal would be to bring the benefits of UPnP into the audio/video world, making the management of content and control of rendering that content format agnostic, while making the technology even easier to use.

It is precisely this problem that UPnP A/V can solve. This chapter introduces UPnP A/V and describes the major functional components and their interactions.

UPnP A/V Architecture Overview

Following the standard UPnP technology model, UPnP A/V uses multiple logical devices with a single control point to coordinate activities between them. This frees the user from having to interact with and configure many distinct devices to make them talk together. Instead, users communicate only with the UPnP A/V control point, which performs the work of discovering and configuring the other UPnP A/V devices on the network.

The basic UPnP A/V architecture uses a triangle of interacting devices, as shown in Figure 17.1.

As shown in Figure 17.1, UPnP A/V defines two device types: the Media Server and the Media Renderer. Simply, the Media Server is a device that is storing the content, while the Media Renderer is the device that renders the content. The UPnP A/V Control Point discovers Media Servers and Media Renders on the network. Using defined UPnP actions, the control point connects a Media Server to the Media Renderer, at which point the two devices can stream the content directly from each other without the UPnP A/V control point getting in the way.

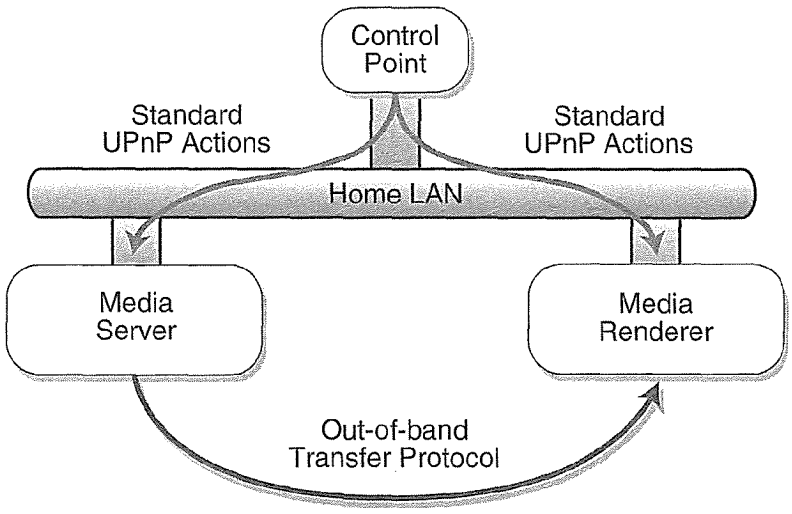


Figure 17.1 UPnP A/V architecture

UPnP A/V was designed to support the following goals:

- Media and transfer protocol agnostic
- Direct source-to-sink transfer of content
- Ability to support A/V devices of all complexities

The first of these goals is satisfied by defining the standard UPnP A/V actions in Media Server and Media Renderer devices that allow the control point to simply act as a matchmaker of sorts between the Media Server and Media Renderer devices, completely independent of the media format or eventual out-of-band transport protocol the devices will use. The three-way architecture of UPnP A/V also allows renderers of content to connect directly to the source of the content, whether that source is in the home network or somewhere on the Internet. The control point, and user for that matter, doesn't need to be bothered with such details. Finally, UPnP A/V supports the third goal of scalable devices of all levels of complexity by defining a minimal set of services and actions that each device must support, which ensures basic interoperability and functionality while also defining a set of optional services and actions that would support an extensive set of advanced device capabilities.

Finally, it's important to note that UPnP A/V supports both the "push" and the "pull" models of streaming content. This means that MediaServers can actively push content to the sink where it will be rendered, or MediaRenderers can actively pull the content from the actual source of the content.

The set of UPnP A/V Specifications available from `http://www.upnp.org` includes:

- UPnP AV Architecture
- Media Server Device Template
- Media Renderer Device Template
- Rendering Control Service Template
- Connection Manager Service Template
- AV Transport Service Template
- Content Directory Service Template

Each specification defines a set of required and optional state variables and actions. The following sections formally introduce the Media Renderer and Media Server device types, along with the set of services and actions each supports.

A_ARG_TYPE

One of the first things you'll notice looking through the A/V specifications is the naming of service state variables with the text "A_ARG_TYPE_" appended to the beginning of the name. Recall that Chapter 13 discussed the requirement that each service action have a corresponding state variable. Yet some device services don't actually need to maintain state for those state variables and, instead, need the defined state variable simply to specify the parameter type.

UPnP A/V solves this problem by distinguishing those state variables that exist only for the purpose of argument typing, with the text "A_ARG_TYPE_" appended to the beginning of the state variable name.

LastChange

UPnP assumes that each device has one set of state variables. When a state variable is defined to support eventing, control points expect to receive event notifications for one instance of that state variable.

Unfortunately, UPnP A/V devices can have multiple instances of the same state variable. Imagine a Media Renderer playing multiple simultaneous audio streams. In this case a set of state variables for each independent stream exists and needs to be evented.

The solution to this problem is a unique state variable called "LastChange." When the value of a state variable changes, information about the change is evented as part of a "LastChange" state variable event. The LastChange state variable events all evented state variables that changed since the last LastChange event was sent.

The format of the event allows devices (and the control points that receive the event) to differentiate between different sets of state variables. Each MediaRenderer has a defined InstanceID for each connection instance that is currently being supported. If a Media Renderer has multiple current connections, it has multiple Instance IDs, each with their own set of state variables describing the current state of that connection.

Figure 17.2 lists the XML for a LastChange event from a device that currently has two instances: one playing video, the other playing audio. Both instances support Volume, but each instance has a different value for Volume.

```
<Event xmlns="urn:schemas-upnp-org:metadata-1-0/AVT_RCS">
  <InstanceID val="0">
    <Brightness val="36">
    <Contranst val="74">
    <Volume val="0">
    ...
  </InstanceID>
  <InstanceID val="1">
    <Mute channel="Master" val="0">
    <Volume val="74">
    ...
  </InstanceID>
</Event>
```

Figure 17.2 LastChange Event XML

UPnP A/V Media Server

The UPnP A/V Media Server device is used by UPnP A/V control points to search for and locate content that is available for playback. Media Servers could support devices such as VCRs, DVD players, CD players, or even live TV tuners. The primary function of a Media Server is to provide a mechanism for UPnP A/V control points to discover what content in the Media Server is available for rendering.

A CD player would enumerate the current CDs inserted in the player, along with the track numbers and other information about the CD. A TV tuner would describe the current list of channels available for viewing, along with detailed program information for each. Finally, a PC could act as a Media Server, advertising all the digital media (MP3s, MPEG videos, and so on) stored on the PC.

Every UPnP A/V Media Server contains a ContentDirectory service, a ConnectionManager service, and an optional AVTransport service. Each of these services is covered in the following sections.

ContentDirectory Service

The ContentDirectory service is probably the most frequently used service in the Media Server. Through the actions defined in the ContentDirectory service, a UPnP A/V control point can browse the list of content available in the Media Server. Content is described in Content Items and Containers that includes available meta-data and other information about the content.

For example, an MP3 file might have a Container that includes not only a reference to the file itself but all the ID3 tag information as well (Artist, Album, Track Number), or even album cover art or song lyrics.

When advertising content, the ContentDirectory service also describes the specific transport protocols available to get the content. These may include HTTP, FTP, RTP, and so on. In this way, when a UPnP A/V control point is trying to match up a Media Server and a Media Renderer, it can make sure there is a common transfer protocol between them that would allow the transferring and rendering of the enumerated content.

The Content Directory Service uses a XML Metadata format called the Digital Item Declaration Language (DIDL). The DIDL defines a standard way to package and describe digital media, such as digital photos. In fact, many digital cameras automatically include DIDL information in the generated photos. Table 17.1 lists the supported state variables for the Content Directory service. Table 17.2 lists required actions for the Content Directory service.

Table 17.1 Content Directory Service State Variables

Name	Required or Optional	Evented	Data Type	Description
TransferID's	O	Yes	string	A CSV list of the currently active Transfers
A_ARG_TYPE_ObjectID	R	No	string	Used with actions that include an ObjectID parameter, it uniquely identifies an individual object within the Content Directory service
A_ARG_TYPE_Result	R	No	string	Used with actions that include a Result parameter, Result returns a DIDL-formatted result string
A_ARG_TYPE_Search Criteria	O	No	string	Used with actions that include a SearchCriteria parameter; provides one or more criteria to be used when querying the Content Directory

Continues

Table 17.1 Content Directory Service State Variables *(Continued)*

Name	Required or Optional	Evented	Data Type	Description
A_ARG_TYPE_Browse Flag	R	No	string	Used with Browse actions, the BrowseFlag parameter specifies options for browsing.
A_ARG_TYPE_Filter	R	No	string	Used with actions that include a Filter parameter; indicates which metadata properties should be returned from a browse or search.
A_ARG_TYPE_Sort Criteria	R	No	string	Used with actions that include a SortCriteria parameter; provides an ordered list of properties to search for
A_ARG_TYPE_Index	R	No	ui4	Used with actions that include an Index parameter; specifies an offset into an arbitrary list of objects
A_ARG_TYPE_Count	R	No	ui4	Used with actions that include a Count parameter; specifies a number of objects
A_ARG_TYPE_UpdateID	R	No	ui4	Used with actions that include an UpdateID parameter; return value is either the SystemUpdateID or the No ContainerUpdateID
A_ARG_TYPE_TransferID	O	No	ui4	Used with actions that include a TransferID parameter; uniquely identifies a specific instantiated file transfer

Continues

Table 17.1 Content Directory Service State Variables *(Continued)*

Name	Required or Optional	Evented	Data Type	Description
A_ARG_TYPE_Transfer Status	O	No	string	Used with actions that include a TransferStatus parameter; provides information about the current status of an instantiated file transfer
A_ARG_TYPE_Transfer Length	O	No	string	Used with actions that include a TransferLength parameter; represents the total length of transfer for a file
A_ARG_TYPE_Transfer Total	O	No	string	Used with actions that include a TransferTotal parameter; represents the total amount of data transferred
A_ARG_TYPE_TagValue List	O	No	string	Contains a CSV list of pairs of XML fragments
A_ARG_TYPE_URI	O	No	URI	Used in actions that include a URI parameter, describes a URI
Search-Capabilities	R	No	string	CSV list of property names used in a search query
Sort-Capabilities	R	No	string	CSV list of tags that can be used to order search or browse results
System-UpdateID	R	Yes	ui4	Changes whenever any object in the Content Directory changes. Used to notify control points when something in the Content Directory has changed
Container-UpdateID's	O	Yes	string	Unordered CSV list of (ContainerID, Containter-UpdateID) pairs used to notify Control Points when containers are updated

Table 17.2 Content Directory Service Actions

Name	Required or Optional	Description
GetSearchCapabilities	R	Returns the supported search capabilities of the Content Directory
GetSortCapabilities	R	Returns a CSV list of meta-data tags that can be used in specifying sort criteria
GetSystemUpdateIDs	R	Returns the current value of the SystemUpdateID state variable
Browse	R	Allows the caller to incrementally browse the native hierarchy of Content Directory Objects.
Search	O	Allows the caller to search the Content Directory for specific objects that match a provided search criteria
CreateObject	O	Creates a new container object
DestroyObject	O	Destroys a previously created container object, including all existing children of the specified container
UpdateObject	O	Modifies, deletes, or inserts metadata into an existing object container
ImportResource	O	Transfers a file from a specified remote source to a specified local destination in the Content Directory service
ExportResource	O	Transfers a file from a local source to a specified remote destination.
StopTransferResource	O	Stops a currently active file transfer
GetTransferProgress	O	Returns the current status of a file transfer
DeleteResource	O	Deletes all elements in the Content Directory Service that match a specified URI
CreateReference	O	Creates a new reference to an existing container object

ConnectionManager Service

The ConnectionManager service is used to create and manage sets of MediaRenderer connections to the MediaServer. Without the ConnectionManager service, a Media Server would be able to support only one active connection at a time. The primary action supported by the ConnectionManager service, `GetProtocolInfo()`, allows an A/V control point to retrieve information about what protocols the device supports sending or receiving.

The optional `PrepareForConnection()` action returns a unique connection InstanceID that UPnP A/V control points can use to reference the created connection when adjusting properties of the connection in the optional AVTransport service, which is defined in the next section. `PrepareForConnection()` also allows an A/V control point to instantiate multiple simultaneous connections to the device. Devices that do not support `PrepareForConnection()` by definition support at most one Connection ID.

The ConnectionManager service is a prime example of how UPnP A/V supports a scalable set of devices, from those simple MediaServers that support only one connection at a time to the fully featured MediaServer that can simultaneously support and manage multiple independent connections.

Table 17.3 lists required state variables for the Connection Manager service. Table 17.4 lists required actions for the Connection Manager Service.

Table 17.3 Connection Manager Service State Variables

Name	Required or Optional	Evented	Data Type	Description
SourceProtocol-Info	R	Yes	string	Contains a CSV list of information about what protocols the Connection Manager supports for sourcing data
SinkProtocol-Info	R	Yes	string	Contains a CSV list of information about what protocols the Connection Manager supports for sinking (rendering)

Continues

Table 17.3 Connection Manager Service State Variables *(Continued)*

Name	Required or Optional	Evented	Data Type	Description
CurrentConnectionIDs	R	Yes	string	CSV list of current active connections, specified by ConnectionID
A_ARG_TYPE_ Connection-Status	O	No	string	Used in actions to provide Connection Status information for a specified ConnectionID
A_ARG_TYPE_ Connection-Manager	R	No	string	Used with actions that wish to specify a "Peer-ConnectionManager"
A_ARG_TYPEDirection	R	No	string	Used with actions that include a Direction parameter, to indicate push vs. pull data transfer
A_ARG_TYPE_ProtocolInfo	R	No	string	Used with actions that include a ProtocolInfo parameter; specifies what protocols the Connection Manager supports
A_ARG_TYPE_ ConnectionID	R	No	i4	Used with actions that include a ConnectionID parameter; indicates a specific Connection ID
A_ARG_TYPE_AV TransportID		No	i4	Used with actions that include a AVTransportID parameter; indicates a specific AVTransportID
A_ARG_TYPE_RcsID	R	No	i4	Used with actions that include a RcsID parameter; specifies a Resource Id

Table 17.4 Connection Manager Service Actions

Name	Required or Optional	Description
GetProtocolInfo	R	Returns information on what protocols the Connection Manager supports
PrepareForConnection	O	Used to allow the device to prepare for and instantiate a new ConnectionID for the purposes of sending or receiving content
ConnectionComplete	O	Used to inform the device that the specified ConnectionID is no longer in use and can be closed
GetCurrentConnectionIDs	R	Returns a CSV list of currently active Connection IDs
GetCurrentConnectionInfo	O	Returns information about a specific Connection ID

AVTransport Service

The optional AVTransport service provides the UPnP A/V control point with the ability to adjust and control the playback of the content stored in the MediaServer. Actions in this optional service include standard VCR-like operations of Play, Pause, Stop, Seek, and so on. Note that the existence of these actions indicates that the MediaServer supports the "pushing" of content to a MediaRenderer, as the control actions are adjusting properties of the stream at its source, not in the Renderer.

For MediaServers that support multiple connections, although only one AVTransport service is advertised, using the different InstanceIDs when invoking actions in effect accesses a different logical instance of the advertised AVTransport service for each different InstanceID.

Table 17.5 lists required state variables for the AVTransport Service. Note that none of the state variables in the AVTransport are evented. Information about all state variable change events are transmitted in the LastChange state variable indexed by the InstanceID.

Table 17.6 lists required actions for the AVTransport Service. All AVTransport actions take an InstanceID as a parameter that determines to which AVTransportInstance the action is applied.

Table 17.5 AVTransport Service State Variables

Name	Required or Optional	Evented	Data Type	Description
TransportState	R	No	string	The current state of the Transport Service; e.g., PLAYING, STOPPED, etc.
TransportStatus	R	No	string	The current error status of the service
PlaybackStorageMedium	R	No	string	Indicates the storage medium of the media specified by the AVTransportURI state variable
RecordStorage-Medium	R	No	string	Indicates the storage medium where the resource specified by AVTrnasportURI is recorded
PossiblePlaybackStorage Media	R	No	string	CSV list of supported storage media that the device can support for playback
PossibleRecord-Storage Media	R	No	string	CSV list of storage media onto which the device can record
CurrentPlay-Mode	R	No	string	Indicates the current play mode (e.g., random, repeat, etc.) of the device.
TransportPlay-Speed	R	No	ui4	Indicates the speed relative to normal speed. (e.g., ∫, π, etc.)
RecordMediumWriteStatus	R	No	ui4	Write protection status of the currently loaded media

Continues

Table 17.5 AVTransport Service State Variables *(Continued)*

Name	Required or Optional	Evented	Data Type	Description
CurrentRecordQualityMode	R	No	ui4	Indicates the current record quality setting
PossibleRecordQuality Modes	R	No	ui4	CSV list of supported record quality modes
NumberOf-Tracks	R	No	string	Current number of tracks available to the AVTransport instance
CurrentTrack	R	No	string	The current track number being played
CurrentTrack-Duration	R	No	string	Duration of the current track
CurrentMedia-Duration	R	No	string	Duration of the current media specified by AVTransportURI
CurrentTrack-MetaData	O	No	URI	DIDL-Lite formatted metadata for the current AVTransportURI
CurrentTrack-URI	R	No	string	URI reference to the current track
AVTransport-URI	R	No	string	URI reference to the current resource controlled by the AVTransport instance
AVTransport-URIMetaData	R	No	string	DIDL-Lite metadata for the resource pointed to by the AVTransportURI state variable
NextAV-TransportURI	R	No	string	Specifies the AVTransportURI to be played when the current AVTransportURI finishes

Continues

Table 17.5 AVTransport Service State Variables *(Continued)*

Name	Required or Optional	Evented	Data Type	Description
NextAV-TransportURI-MetaData	R	No	string	DIDL-Lite metadata for the resource pointed to by the NextAVTransport-URI state variable
RelativeTime-Position	R	No	string	Current time position from the beginning of the current track
AbsoluteTime-Position	R	No	string	Current time position from the beginning of the current specified media
RelativeCounterPosition	R	No	i4	Current position of a counter from the beginning of the current track
AbsoluteCounterPosition	R	No	i4	Current position of a counter from the beginning of the current media
Current-Transport-Actions	O	No	string	CSV list of the current supported control actions in the AVTransport service
A_ARG_TYPE_SeekMode	R	No	string	Used in actions that have a SeekMode parameter; indicates the allowed units to seek
A_ARG_TYPE_SeekTarget	R	No	string	Used in actions that have a SeekTarget parameter; indicates a target track number to which the disk can seek
A_ARG_TYPE_InstanceID	R	No	ui4	Used in actions that have a InstanceID parameter; indicates the specific InstanceID to which the action should apply

Table 17.6 AVTransport Service Actions

Name	Required or Optional	Description
SetAVTransportURI	R	Sets the AVTransportURI state variable
SetNextAVTransportURI	O	Sets the NextAVTransportURI state variable
GetMediaInfo	R	Returns information associated with the current media
GetTransportInfo	R	Returns information associated with the current transport state
GetPositionInfo	R	Returns information about the current position of the resource specified by the AVTransportURI state variable
GetDeviceCapabilities	R	Returns information about device capabilities for the current resource specified by the AVTransportURI state variable
GetTransportSettings	R	Returns information on AVTransport settings for the specified InstanceID, (current play mode, recording quality mode, etc.)
Stop	R	Stop the current resource specified by the AVTransportURI state variable
Play	R	Play the current resource specified by the AVTransportURI state variable
Pause	O	Pause the current resource specified by the AVTransportURI state variable
Record	O	Start recording on the specified transport instance
Seek	R	Seek forward or reverse to the specified seek target on the current resource
Next	R	Advance to the next track in the resource specified by the AVTransportURI.; only applies to playlists
Previous	R	Advance to the previous track in the resource specified by the AVTransportURI; only applies to playlists

Continues

Table 17.6 AVTransport Service Actions *(Continued)*

Name	Required or Optional	Description
SetPlayMode	O	Set the desired play mode (random, repeat, etc.) of the device
SetRecordQualityMode	O	Set the desired record quality mode for the device
GetCurrentTransport Actions	O	CSV list of current list of supported required and optional actions in the AVTransport Service instance

UPnP A/V Media Renderer

The other UPnP device that makes up the UPnP A/V triangle, a UPnP A/V Media Renderer, is used to play back, or render, content from the network. Media Renderers could include televisions, stereos, in-home speakers, or even a wild piece of digital art that reacts to music. The Media Renderer is the end-point device that the user has selected to render some content. As such, it has services and actions that support controlling properties of how the content is rendered, controlled, and set up. The next few sections introduce the required ConnectionManager, RenderingControl, and optional AVTransport services.

ConnectionManager Service

Similar to the Connection Manager service provided by the Media Server, the Media Renderers Connection Manager service allows UPnP A/V control points the opportunity to query the device and find out what types of media and transport protocols the device supports when rendering content.

Similar to Media Servers that support multiple simultaneous connections, Media Renderers can also support such a capability through the Connection Manager service. Think of an A/V receiver with many different output channels for different rooms in the house. Certainly this kind of Media Renderer would benefit from the creation of different independent connection instances. The InstanceID returned from the `PrepareForConnection()` call is used in future action invocations in the other services supported by the Media Renderer. The Connection Manager service also provides actions for control points to discover how many InstanceIDs are active, as well as retrieving properties of each active connection.

Like Media Servers that support only one connection, Media Renderers that are capable of rendering only one stream at a time return an InstanceID of 0. State variables and actions supported in the Connection Manager service are listed in Tables 17.3 and 17.4 respectively.

RenderingControl Service

The heart and soul of the Media Renderer, the RenderingContol service provides UPnP A/V control points the ability to adjust properties of how content is being rendered. Using the actions of the RenderingControl service, properties such as Volume level, Brightness, Loudness, Tint, Color Contrast, and so on can be controlled by users of the Media Renderer. Note that not all of these actions are required. Certainly it doesn't make sense for a audio-only Renderer to support adjusting the brightness levels of a display.

Table 17.7 lists required state variables for the RenderingControl service. Like the AVTransport service, the RenderingControl service does not individually event any of the supported state variables. It instead relies upon the LastChange state variable to package evented state variables ordered by InstanceID. Table 17.8 lists required actions for the RenderingControl service.

Table 17.7 Rendering Control Service State Variables

Name	Required or Optional	Evented	Data Type	Description
PresetNameList	R	No	string	CSV list of valid present names currently supported by the device
Brightness	O	No	ui2	Current brightness setting of the display device
Contrast	O	No	ui2	Current contrast setting of the display device
Sharpness	O	No	ui2	Current sharpness setting of the display device
RedVideoGain	O	No	ui2	Current setting of the red gain control for the display device

Continues

Table 17.7 Rendering Control Service State Variables *(Continued)*

Name	Required or Optional	Evented	Data Type	Description
GreenVideo-Gain	O	No	ui2	Current setting of the green gain control for the display device
BlueVideoGain	O	No	ui2	Current setting of the blue gain control for the display device
RedVideoBlackLevel	O	No	ui2	Current setting for the minimum output intensity of red for the display device
GreenVideo-BlackLevel	O	No	ui2	Current setting for the minimum output intensity of green for the display device
BlueVideoBlackLevel	O	No	ui2	Current setting for the minimum output intensity of blue for the display device
Color-Temperature	O	No	ui2	Current setting for the color quality of white for the display device
Horizontal-Keystone	O	No	i2	Current level of compensation for horizontal distortion in the display image
Vertical-Keystone	O	No	i2	Current level of compensation for vertical distortion in the display image
Mute	O	No	boolean	Boolean value designating whether the device is currently muted
Volume	O	No	ui2	Current volume setting of the audio channel

Continues

Table 17.7 Rendering Control Service State Variables *(Continued)*

Name	Required or Optional	Evented	Data Type	Description
VolumeDB	O	No	i2	Current volume setting of the audio channel in 1/256 of a decibel (dB)
Loudness	O	No	boolean	Boolean value designating whether loudness is active
A_ARG_TYPE_Channel	R	No	string	Used in actions to specify a specific audio or video channel
A_ARG_TYPE_InstanceID	R	No	ui4	Used in actions to specify a InstanceID for the action to apply to
A_ARG_TYPE_PresetName	R	No	string	Used in actions to specify the name of a device preset

Table 17.8 Rendering Control Service Actions

Name	Required or Optional	Description
ListPresets	R	Returns the list of currently defined presets
SelectPreset	R	Restores Rendering Control state variables to a defined preset set of values
GetBrightness	O	Returns the current value of the Brightness state variable
SetBrightness	O	Sets the value of the Brightness state variable
GetContrast	O	Returns the current value of the Contrast state variable
SetContrast	O	Sets the value of the Contrast state variable
GetSharpness	O	Returns the current value of the Sharpness state variable

Continues

Table 17.8 Rendering Control Service Actions *(Continued)*

Name	Required or Optional	Description
SetSharpness	O	Set the value of the Sharpness state variable
GetRedVideoGain	O	Returns the current value of the RedVideoGain state variable
SetRedVideoGain	O	Set the value of the RedVideoGain state variable
GetGreenVideoGain	O	Returns the current value of the GreenVideoGain state variable
SetGreenVideoGain	O	Set the value of the GreenVideoGain state variable
GetBlueVideoGain	O	Returns the current value of the BlueVideoGain state variable
SetBlueVideoGain	O	Set the value of the BlueVideoGain state variable
GetRedVideoBlackLevel	O	Returns the current value of the RedVideoBlackLevel state variable
SetRedVideoBlackLevel	O	Set the value of the RedVideoBlackLevel state variable
GetGreenVideoBlackLevel	O	Returns the current value of the GreenVideoBlackLevel state variable
SetGreenVideoBlackLevel	O	Set the value of the GreenVideoBlackLevel state variable
GetBlueVideoBlackLevel	O	Returns the current value of the BlueVideoBlackLevel state variable
SetBlueVideoBlackLevel	O	Set the value of the BlueVideoBlackLevel state variable
GetColorTemperature	O	Returns the current value of the ColorTemperature state variable
SetColorTemperature	O	Set the value of the ColorTemperature state variable

Continues

Table 17.8 Rendering Control Service Actions *(Continued)*

Name	Required or Optional	Description
GetHorizontalKeystone	O	Returns the current value of the HorizontalKeystone state variable
SetHorizontalKeystone	O	Set the value of the HorizontalKeystone state variable
GetMute	O	Returns the current value of the Mute state variable
SetMute	O	Sets the value of the Mute state variable
GetVolume	O	Returns the current value of the Volume state variable
SetVolume	O	Set the value of the Volume state variable
GetVolumeDB	O	Returns the current value of the VolumeDB state variable
SetVolumeDB	O	Set the value of the VolumeDB state variable
GetVolumeDBRange	O	Returns the valid range for the VolumeDB state variable
GetLoudness	O	Returns the current value of the Loudness state variable
SetLoudness	O	Set the value of the Loudness state variable

AVTransport Service

Just like the UPnP A/V Media Server, the Media Renderer can also optionally support the AVTransport service. The AVTransport service provides control points with the ability to use VCR-like operations (Play, Pause, Stop, and so on) to control the playback of content on the Renderer.

Supporting the AVTransport service on a Media Renderer device typically means that the Renderer does most of the active streaming from the source of the content, in a "pull" method. In this way, calling actions in the AVTransport service on the Media Renderer device would cause it to adjust the rate or specific manner in which it is pulling, or streaming, the content from its source.

Service state variables and actions for the AVTransport service are defined in Tables 17.5 and 17.6 respectively.

UPnP A/V Control Point

Perhaps the most important node on the UPnP A/V network, the UPnP A/V Control Point, performs all discovery and coordination between Media Servers and Media Renderers. Recall that the UPnP architecture provides no mechanism for UPnP devices to communicate directly with each other. This is the role that the A/V control point plays. When the user wants the Renderer to pause playing an audio stream, it is the A/V control point that invokes the action on the user's behalf.

Typically, the application the user is interacting directly with is the A/V control point. When users want to search for content, it is the A/V control point that discovers all the Media Servers on the network and invokes actions in each, providing the search criteria specified by the user.

Often the unsung hero in a UPnP A/V network, all the intelligence to use Media Servers and Media Renderers is in the A/V control point. Recall that there may be multiple different Media Servers and Renderers on the network. A common feature for A/V control points is to aggregate the list of available content in the dozen or so Media Servers that may exist on the network into a single list viewable by the user. In this way, the A/V control point can take much of the complexity of the home network's topology out of the picture. Users don't care exactly where on what device content is stored, they're interested simply in what is available for them to view.

Similar to setting up a graph of nodes, the A/V control point must work to find an appropriate renderer (often based on user input if there are several renderers in the home) to play back the content the user has selected based upon what format types and transfer protocols are supported by each Media Server and Media Renderer.

The power of UPnP A/V is that all of this complicated activity is completely abstracted to the end user. No longer does the user have to manually configure a playback device or figure out what format of video may exist on a given CD, and which of the playback devices in the home can support it.

A home of UPnP A/V devices is where users can conveniently and quickly access all of their digital media without having to worry about complicated setup and configuration of Renderer devices.

End User Scenario: Audio Playback

In this section, you'll go through the process of selecting some content to play on a given Media Renderer, highlighting the specific UPnP device actions invoked at each step. You'll assume that you have a single Media Server and a single Media Renderer, that the Media Renderer supports the "pull" model of distribution using HTTP, and that it supports the playback of MP3 audio files.

To discover content, the A/V control point must discover and invoke a `Browse()` or `Search()` action in the Media Server's Content Directory service to retrieve a container object for the content you'd like to play back. Next, the A/V control point must discover and invoke a `GetProtocolInfo()` action in the Media Renderer to match appropriate protocol formats between the content you want to play and what the Media Renderer supports.

Then the A/V control point calls `PrepareForConnection()` in both the Media Server and Media Renderer (if implemented) storing the returned InstanceIDs. At this point the A/V control point can invoke `Play()` on the Media Renderer, which causes the Renderer to initiate an out-of-band connection to the device to begin streaming the content to the Media Rendering device. Once playing, the A/V control point can now invoke actions in the Rendering Control service to adjust the properties of the rendering stream, such as volume, brightness, and so on. Figure 17.3 is a sequence diagram of this process.

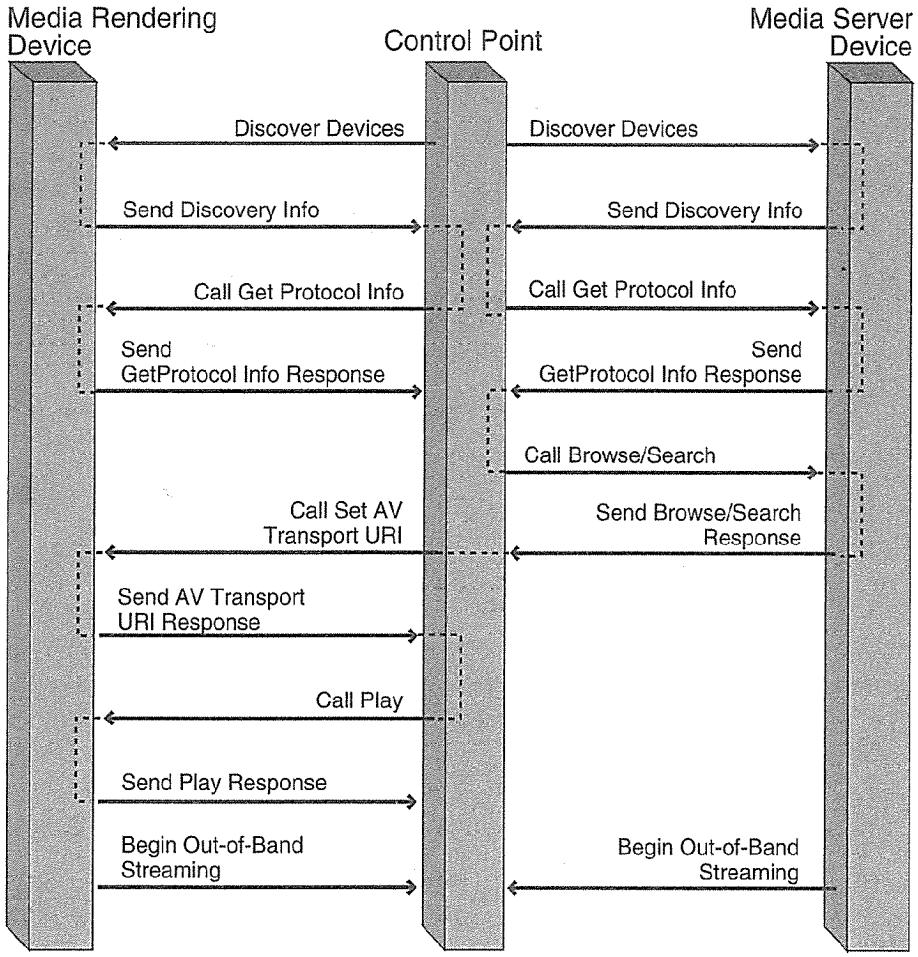


Figure 17.3 Playback Message Sequence Diagram

Summary

This chapter introduced a powerful new architecture for supporting the management and playback of media using UPnP. It began with an overview of UPnP A/V's unique three-way interaction model and continued with coverage of each of the distinct device types in a UPnP A/V network. It finished with a detailed discussion of the real-world scenario of playing back audio on a Media Rendering device. In the next chapter you'll use the information you learned in this chapter to actually implement basic support for a UPnP A/V Media Rendering device in your Super Toaster.

Glossary

Action A command presented by a service that can be invoked by control points. A service typically has many actions. Each action has a set of optional input and output parameters and an optional return value.

Ad-Hoc Network A network where there are no preexisting infrastructure devices and services (such as a DHCP server) and the network nodes themselves make up the network. Devices use other means, such as Auto-IP, to acquire IP addresses, and are then able to communicate.

Address Resolution Protocol (ARP) A network protocol that maps a network level address, such as an IP address, to its corresponding data link address, such as an Ethernet addresses. UPnP devices use ARP to find out whether an address selected with Auto-IP is currently used by any other devices.

Addressing The process by which a UPnP device acquires and releases its address. Addressing is the first step in UPnP networking—a device must acquire an address before it can advertise itself.

Administrative Scope A way to limit the reach of multicast data. A special range of IP multicast addresses, 239.0.0.0 to 239.255.255.255, is called the administratively scoped IPv4 multicast address space.

Within this range, addresses are partitioned to have predefined semantics about how broadly the data will propagate. Administrative scoping provides a simple way to contain IP multicast communication within the administrative boundaries of an organization.

Advertisement A message from a device to control points announcing that the device is now available.

All Hosts Group A particular multicast IP address, 224.0.0.1, used to address all of the multicast hosts that are directly connected to the same network as the sender.

Argument A parameter for an action exposed by a service. Each argument may be an input argument or an output argument, but not both. One of the output arguments may be designated as the action's return value.

Auto-IP A method by which an endpoint on an IP network automatically chooses an IP address and subnet mask in the absence of a central service, such as a DHCP, to manage addresses. This method is described in the Internet Draft, the "Dynamic Configuration of IPv4 Link-Local Addresses."

Chunked (Transfer) Encoding A method used by HTTP servers that breaks the response into smaller chunks and sends them in a series. Such responses are identified by including the `Transfer-Encoding` header with the value set to `chunked`.

Control The method invocation process where control points invoke methods provided by a device's services. The control protocol used between UPnP control points and devices is the Simple Object Access Protocol (SOAP).

ControlURL Each service listed in a device description document has an element, the `<controlURL>`, that provides the URL where all control messages for that service are to be sent.

Control Point An entity on the network that invokes the functionality provided by a UPnP device. The control point may discover devices, retrieve device and service descriptions, invoke actions on services, query for state variables, and receive events from services.

Cookie User-specific information stored on a client computer by a web site so that the information can be passed to the server on future requests. Cookies are often used by stateless protocols, such as HTTP, to maintain state on the client to pass to the server with each request.

Decentralized Discovery An approach to service discovery whereby there is no central store to maintain information about resources, their location, and their availability. Instead, each client directly queries the network and each resource responds directly to these requests.

Description A phase of UPnP device operation that allows devices to present information about themselves and the services that they provide, in the form of XML-based device and service description documents, to control points on the network.

Device A logical container for services. A device can acquire an address, can be discovered, can provide information about itself in the form of a device description document, and can provide a presentation page. Control actions and eventing work directly with services, however. A device may contain other embedded devices.

Device Control Protocol (DCP) *See* device description document.

Device Description (Document) A document, expressed in the XML-based UPnP Template Language, provided by UPnP devices that contains information about the device (such as manufacturer name, model name, serial number, and so on), a list of services provided by the device, and a list of any embedded devices. Control points retrieve the device description document to learn about the device and its services. UPnP device vendors fill in placeholders in a device template standardized by one of the working committees.

Device Type A formal definition of a logical device, as expressed in a device description document. Standard device types are defined by working committees of the UPnP Forum.

Discovery A phase of UPnP device operation that allows control points to search for devices and services on the network and find ones that meet its search criteria.

Document Object Model (DOM) An API for HTML and XML documents that defines the object model for these documents. The W3C

has specified the DOM API in a language-independent way and has been implemented in many different programming languages. With the DOM programmers can access, change, delete, or add just about anything found in HTML or XML documents.

Dynamic Host Configuration Protocol (DHCP) A client/server protocol that provides a framework for passing configuration information to hosts on a TCP/IP network, including the host's IP address, subnet mask, default gateway, and domain name server.

Embedded Device A device logically contained within another device. Any embedded devices are listed in a device's description document.

Event A notification message sent by a service to control points that indicates a change in one or more of the service's state variables.

Event Key A value maintained by the publisher of event messages for each subscriber as an error detection mechanism to ensure that subscribers have received all event messages sent.

EventSubURL Each service listed in a device's description document has an element, the <eventSubURL>, that provides the URL where control points can register to receive events from the service. The GENA protocol is used to manage subscriptions and send event messages.

Evented State Variable A service will send event notifications to control points when this state variable changes. State variables are marked as evented or not in the service's description document.

Eventing The phase of UPnP device operation where services send notifications of changes to state variables to control points.

General Event Notification Architecture (GENA) A protocol that implements a publisher/subscriber system whereby a subscriber may request, renew, or cancel a subscription. Event notification messages are sent from the publisher to subscribers.

Host Group A group of endpoints on an IP network that receive multicast data from a sender. Each host in the logical group shares a common multicast address and receives any data sent to the multicast address. The membership in this logical group can change over time.

HTTPMU HTTP over multicast UDP. HTTPMU allows sending HTTP messages to many recipients simultaneously. HTTPMU enables a group communication model using HTTP-style request/response messages.

HTTPU HTTP over unicast UDP. With HTTPU, a host can send an HTTP-formatted message to another host without the expense of setting up a TCP connection.

HTTP Resource A network-based service that is accessed via the HTTP protocol. The SSDP protocol is used to discover HTTP resources.

Initial Event Message A special event message sent when a control point first subscribes to receive events from a service. This special first message includes the names and values for all evented variables provided by the service and allows the subscriber to initialize its model of the state of the service.

Initial Request Line The first line of an HTTP request that includes the HTTP method, the path of the requested resource, and the version of HTTP being used.

Infrastructure Network *See* Managed Network.

Internet Gateway Device (IGD) The first device standard produced by one of the UPnP Forum working committees. The IGD supports sharing of Internet connections, advanced connection-management features, management of host-configuration services, and support for transparent Internet access.

Lease (of an IP address) An agreement, between a server managing IP addresses and a client acquiring an IP address, that the client may use the IP address for a limited period of time. The server assigns the address to the client. Once the lease expires, the server may assign the address to another client.

Managed Network A network supported by devices and services, such as a DHCP server, dedicated to the operation of the network (as opposed to an ad-hoc network that has no such supporting devices or services).

Marketing Committee A committee of the UPnP Forum that undertakes joint member promotion of UPnP, including representing the UPnP Forum at industry trade shows.

Moderation of Events An extension to the UPnP Template Language that specifies a limit for the rate at which events are sent from a publisher to subscribers. It is used for state variables that would otherwise change too rapidly for eventing to be useful.

Notification *See* Event.

Phases of UPnP The sequence of operation of UPnP devices consisting of addressing, description, discovery, control, eventing, and presentation.

Presence Announcement In SSDP, resources announce their presence on the network, letting potential clients know of their availability.

Presentation A phase of UPnP device operation that allows devices to provide a Web page for manual control and administration.

Presentation Page UPnP Devices can use their embedded web servers to provide a web interface for manual management and control of the device. An administrator using a web browser can load the device's Presentation URL and view information about the device and control it.

Publisher In a Publisher/Subscriber system, the publisher is a source of event messages delivered to subscribers, who register to receive them. For UPnP devices, the services can publish state change events to control points.

Publisher/Subscriber Model A software design pattern used to implement event notification. In this model, the publisher is the source of events and grants a client a subscription when it registers interest in receiving events provided by the publisher. Upon the occurrence of an event, the publisher delivers an event notification to the subscriber.

QueryStateVariable A predefined action implemented by every service to provide access to its state variables.

Related State Variable Every input argument to an action is associated with one of the service's state variables—its related state variable.

Return Value The output argument to an action that has been designated as returning the result of the action. Only one of an action's arguments may be its return value.

Root Device A logical device that is not embedded in any other logical device.

SCPD *See* UPnP Service Template.

SCPDURL The URL for a service's description document. Each service listed in a device description document has an SCPDURL pointing to the service's description document.

Service The basic unit of functionality provided by UPnP devices. A service provides actions that can be invoked by control points and state variables that can be used to model the state of an underlying physical device.

Service Description Document An XML document expressed in the UPnP Template language that provides the formal definition of a logical service. Standard service description templates are defined by the working committees of the UPnP Forum and are filled in by device vendors.

ServiceId A sub-element of a service element in a device description document, the `<serviceId>` uniquely identifies a service. For standard services defined by a UPnP Forum working committee, the serviceId begins with `urn:upnp-org:serviceId:` followed by a service ID suffix.

Service Type A service type is a URI that identifies the type, or function, of a particular resource. SSDP provides the mechanisms for discovering resources by service type. Service types for UPnP devices and services are defined by UPnP working committees for each standard device type. Standard service types are denoted by `urn:schemas-upnp-org:service:` followed by a unique name assigned by the working committee, a colon, and an integer version number.

Simple Object Access Protocol(SOAP) A protocol that brings together XML and HTTP to provide a Web-based messaging and remote procedure call mechanism. XML is used to express the contents of the messages, while HTTP is used to send the messages to their destination.

Simple Service Discovery Protocol (SSDP) A protocol for discovery of HTTP-based resources on the local area network that doesn't require any configuration, management, or administration.

ssdp:alive A message from a UPnP device joining the network that advertises the availability of one of its devices or services. When joining the network, the device advertises all of the devices and services it is providing. The `ssdp:alive` message is sent using a GENA `NOTIFY` method over the SSDP multicast channel.

ssdp:bye-bye A message sent by a device being removed from the network that notifies control points that the device or one of its services are no longer available. One `ssdp:bye-bye` message is sent for each `ssdp:alive` advertisement previously sent out by the device.

State Change (Event) A notification, sent from a service to a control point, that one or more of its evented state variables has changed.

State Table A service's state variables.

State Variable A variable, maintained by a service, that has a name, type, optional default value, optional constraint values, and which may trigger events when its value changes. State variables may be used to model the state of a physical device.

Steering Committee The high-level directing body of the UPnP Forum. The Steering Committee provides business leadership and makes decisions for the UPnP Forum. As the organization's management team, the Steering Committee oversees the working committees for defining device descriptions.

Subscriber In a publisher/subscriber software design pattern, the subscriber is the recipient of event messages sent by the publisher. In UPnP, control points can subscribe to state change events from services.

Subscriber List A list of registered subscribers in a system that implements a publisher/subscriber software design pattern.

Subscription An agreement between a publisher and a subscriber in a publisher/subscriber design that the subscriber will receive requested events from the publisher.

Subscription ID An identifier, generated by the publisher, that the subscriber presents when referencing a subscription. For example, the subscriber might present the subscription id when renewing or canceling a its subscription.

Subscription Cancellation A message, sent by a subscriber, such as a UPnP control point, to a publisher, such as a UPnP service, to discontinue its subscription for events.

Subscription Expiration The end of a time-based agreement between a publisher and a subscriber. When a subscription expires, the subscriber is removed from the subscription list and the publisher stops sending events to the subscriber.

Subscription Renewal A message, sent by a subscriber to the publisher, to keep a current subscription active, perhaps by extending the time of subscription expiration.

Technical Committee A group in the UPnP Forum that consists of technical representatives from various companies. The technical committee handles technical issues from working committees. They are responsible for the technical "big picture."

Time to Live (TTL) Field A field of the IPv4 header that controls the number of times, or "hops," that an IP datagram is allowed to traverse a router. Changing the TTL value at the source of communication will extend or contract the reach of the IP datagram.

Uniform Resource Identifier (URI) An identifier that provides a conceptual mapping from an identifier to a particular resource on the Web. For example, a particular URI might reference the home page for a particular organization whose contents changes over time, while the URI for the page remains the same. URIs are further classified as Uniform Resource Locators and Uniform Resource Names.

Uniform Resource Locator (URL) A URI that identifies a resource by specifying its location rather than identifying the resource by name or other attribute. Many URL schemes are named after protocols, such as http, ftp, and gopher.

Uniform Resource Name (URN) A label for a resource that is required to remain globally unique and persistent even when the resource ceases to exist or becomes unavailable.

Unique Service Name (USN) An SSDP header and associated value provided in SSDP messages that uniquely identifies an SSDP service.

UPnP Device Architecture A document that defines the protocols and conventions for communication between UPnP control points and devices.

UPnP Device Template An XML document derived from the UPnP Template Language that lists what a particular device type must include in its device description document, including device type, required embedded devices (if any), and required services. Standard UPnP device templates are defined by UPnP Forum working committees. The device template is completed by a device vendor.

UPnP Forum A cross-industry group created to guide the creation of the UPnP standards. The UPnP Forum consists of more than 550 companies, including industry leaders in consumer electronics, computing, home automation, home security, appliances, printing, photography, computer networking, and mobile products.

UPnP Implementer's Corporation (UIC). An independent company created by the UPnP Steering Committee, the UIC manages the conformance testing of devices to UPnP standards, administers the UPnP certification process, licenses tests to UIC members, reviews the manages the test results, and issues certificates of conformity to devices that pass the tests.

UPnP Service Template An XML document derived from the UPnP Template Language that lists what a particular service type must include in its service description document, including actions, parameters, and state variables. Standard UPnP service templates are defined by UPnP Forum working committees. The service template is completed by a device vendor.

UPnP Stack The set of protocols used by UPnP devices.

UPnP Template Language An XML schema that defines the elements and attributes used in UPnP Device and Service Templates. The UPnP Template Language is defined by the UPnP Device Architecture.

Working Committee An organizational elements of the UPnP Forum, formed as needed by participants to define standard device types.

XML A meta-language with which to develop markup languages that specify the structure of data and how various elements of the data relate. XML is becoming the de facto Internet standard for the representation of information.

Zero Configuration Network A network where the user is not required to configure devices before they are used on the network.