Internet Core Protocols
The Definitive Guide

Eric A. Hall

O'REILLY®

Beijing · Cambridge · Farnham · Köln · Paris · Sebastopol · Taipei · Tokyo

1

An Introduction to TCP/IP

If you've been using TCP/IP-based networking products for any length of time at all, you're probably already aware of how IP addressing, routing, and other fundamental aspects of the Internet family of protocols work, at least from a user's perspective.

What you probably don't know—unless you've been formally trained in these subjects—is what makes TCP/IP work from the wire's perspective, or from the perspective of the applications in use on your network. This chapter provides you with an introduction to these viewpoints, providing you with a better understanding of the nature of the traffic on your network.

A Brief History of the Internet

Before you can understand how TCP/IP works—or why it works the way it does—you first have to understand the origins of the networking protocols and the history of the Internet. These subjects provide a foundation for understanding the basic design principles behind TCP/IP, which in turn dictate how it is used today.

TCP/IP presented a radical departure from the traditional computer networking services in use during its development. In the early days of commercial computing (the late 1960s), most companies bought a single large system for all of their data processing needs. These systems used proprietary networking architectures and protocols, which primarily consisted of plugging dumb terminals or line printers into an intelligent communications controller, each of which used proprietary networking protocols to communicate with the central hosts.

Most of the early computer networks used this hierarchical design for their proprietary network protocols and services. As users' computing requirements expanded, they rarely bought a different system from a different vendor, but instead added new components to their existing platforms or replaced the existing system with a newer, larger model. Cross-platform connectivity was essentially unheard of, and was not expected. To this day, you still can't plug an IBM terminal into a DEC system and expect it to work. The protocols in use by those devices are completely different from each other.

As the use of computers became more critical to national defense, it became clear to the U.S. military in particular that major research centers and institutions needed to be able to share their computing resources cooperatively, allowing research projects and supercomputers to be shared across organizational boundaries. Yet, since each site had different systems (and therefore different networking technologies) that were incompatible with the others, it was not possible for users at one site to use another organization's computing services easily. Nor could programs easily be ported to run on these different systems, as each of them had different languages, hardware, and network devices.

In an effort to increase the sharing of resources, the Advanced Research Projects Agency (ARPA) of the Department of Defense (DOD) began coordinating the development of a vendor-independent network to tie the major research sites together. The need for a vendor-independent network was the first priority, since each facility used different computers with proprietary networking technology. In 1968, work began on a private packet-switched network, which eventually became known as ARPAnet.

ARPAnet was the world's first wide-area packet-switching network, designed to allow individual units of data to be routed across the country as independent entities. Previous networks had been circuit-switched, involving dedicated end-to-end connections between two specific sites. In contrast, the ARPAnet allowed organizations to interconnect into a mesh-like topology, allowing data to be sent from one site to another using a variety of different routes. This design was chosen for its resilience and built-in fault-tolerance: if any one organization were bombed or otherwise removed from the network, it wouldn't affect the rest of the organizations on the network.

During this same time period, other network providers also began interconnecting with the ARPAnet sites, and when these various networks began connecting to each other, the term "Internet" came into use. Over the next few years, more organizations were added to the ARPAnet, while other networks were also being developed, and new network technologies such as Ethernet were beginning to gain popularity as well.

All of this led to the conclusion that networking should be handled at a higher layer than was allowed by the ARPAnet's packet-switching topology. It became increasingly important to allow for the exchange of data across different physical networks, and this meant moving to a set of networking protocols that could be implemented in software on top of any physical topology, whether that be a packet-switched WAN such as ARPAnet or a local area network (LAN) topology such as Ethernet.

TCP/IP to the Rescue

In 1973, work began on the TCP/IP protocol suite, a software-based set of networking protocols that allowed any system to connect to any other system, using any network topology. By 1978, IP version 4 (the same version that we use today) had been completed, although it would be another four years before the transition away from ARPAnet to IP would begin. Shortly thereafter, the University of California at Berkeley also began bundling TCP/IP with their freely distributed version of Unix, which was a widely used operating system in the research community.

The introduction and wide-scale deployment of TCP/IP represented a major ground-shift in computer networking. Until the introduction of TCP/IP, almost every other network topology required that hardware-based network nodes send traffic to a central host for processing, with the central host delivering the data to the destination node on behalf of the sender. For example, Figure 1-1 shows a host-centric networking architecture. In this model, devices are attached to a centralized system that coordinates all network traffic. A user at a terminal could not even send a screen of text to a printer without first sending the data to the central host, which would parse the data and eventually send it to the printer for printing.

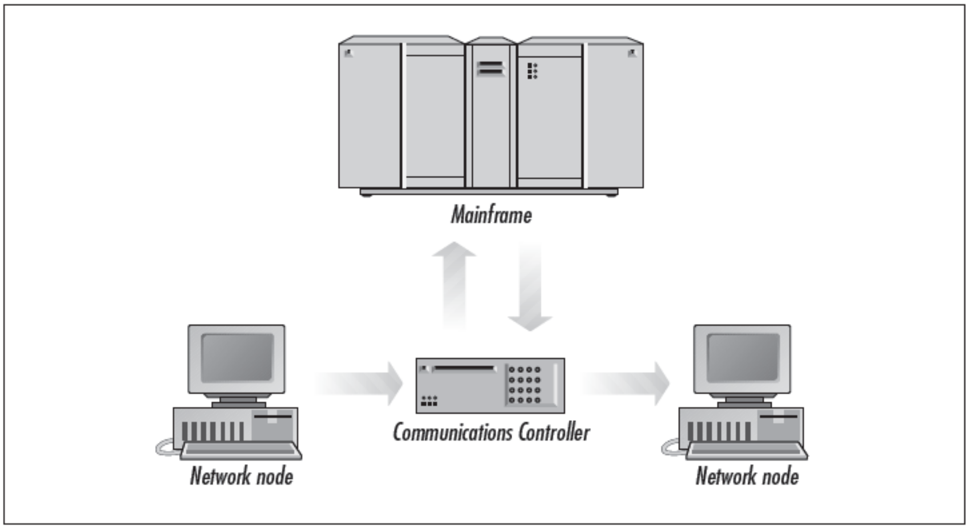


Figure 1-1. Host-centric networking

ROKU EXH. 1002

But with TCP/IP, each network device was treated as a fully functional, self-aware network end-point, capable of communicating with any other device directly, without having to talk to a central host first. IP networks are almost anarchic, with every device acting as a self-aware, autonomous unit, responsible for its own network services, as illustrated in Figure 1-2.

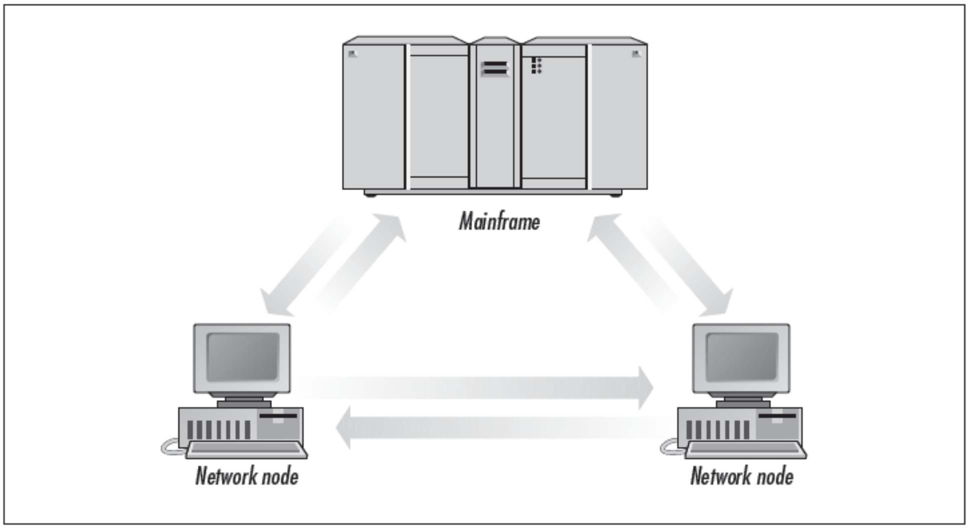


Figure 1-2. Node-centric networking

This design allowed for application- and resource-sharing on a national scale, since a top-down model simply would not work with millions of widely distributed devices. In addition, this design also provided reliability in case any part of the network was damaged, since a host-based model would simply stop functioning if the central host was destroyed or disabled.

The Internet Today

Over time, the ARPAnet evolved into an open "network-of-networks" using TCP/IP, with educational, commercial, and other organizations connected to each other through an interwoven mesh of networks. Today this type of mesh architecture is far less common, replaced by a much more structured hierarchy.

Rather than organizations connecting to each other directly, most organizations now connect to a local network access provider who routes network traffic upwards and outwards to other end-point networks.

Generally speaking, there are only a handful of top-level Internet Service Providers (ISPs), each of which provide major interconnection services around the country or globe. Most of these firms are telecommunications companies that specialize in large-scale networking (such as long-distance providers like MCI WorldCom and Sprint).

Below these top-level carriers are local or regional access providers who offer regional access and lower-speed connection services to end users directly (these mid-level carriers are sometimes referred to as Internet Access Providers, or "IAPs"). This design is represented in Figure 1-3.

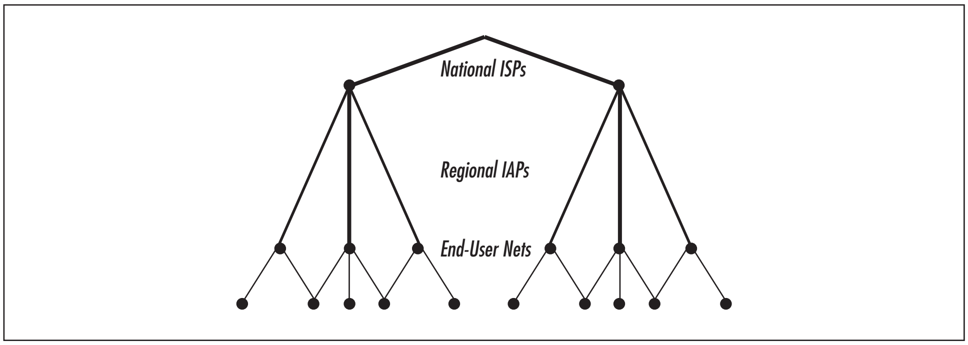


Figure 1-3. The hierarchical architecture of the Internet

Visually, the Internet can be thought of as a few major networking companies who provide large-scale "backbone" services around the world, followed by a large number of secondary providers that resell bandwidth on those networks. At the end of the line are the end-leaf organizations that actually generate the traffic that crosses these networks.

The Internet, Defined

Simply having a lot of interconnected networks does not by itself mean that you have the "Internet." To "internet" (with a lowercase "i") means to interconnect networks. You can create an internet of Macintosh networks using AppleTalk and some routers, for example. The term "Internet" (with a capital "I") refers to the specific global network of TCP/IP-based systems, originally consisting of ARPAnet and the other research networks.

There have been lots of private and public networks that have offered a multi-layer network design (private SNA* networks from the 1980s are a good example of this). Therefore, the Internet in particular is a collection of networks that support host-to-host communications using TCP/IP protocols.

* SNA stands for Systems Network Architecture, a proprietary IBM networking protocol.

Under this definition, the network is made up of intelligent end-point systems that are self-deterministic, allowing each end-point system to communicate with any host it chooses. Rather than being a network where communications are controlled by a central authority (as found in many private networks), the Internet is specifically meant to be a collection of autonomous hosts that can communicate with each other freely.

This is an important distinction, and one that is often overlooked. For example, many of the private networks have offered mail-delivery services for their customers, allowing a user on one network to send email to another user on another network, but only by going through a predefined mail gateway service. Conversely, the Internet allows users to exchange mail directly, without going through a central politburo first. In this regard, the Internet is a collection of self-deterministic, autonomous hosts.

Having hosts communicate with each other directly is not enough to make the Internet, however. Many networks have offered users the ability to communicate directly with other hosts on those networks, and those networks have not been considered as parts of the Internet per se. For example, there have been many private DECnet networks that have offered this capability, and Novell offers a similar service using IPX today.

The last key criteria is that the Internet is a collection of networks that allows host-to-host communications through voluntary adherence to open protocols and procedures defined by Internet standards. Therefore, in order for these networks to be parts of the Internet, they must also use Internet protocols and standards, allowing for vendor-neutral networking.

This is perhaps the most important part of the entire definition, since the use of consistent protocols and services is what allows the Internet to function at all. For example, it is not enough for a private network to allow users to send email messages to each other directly. Rather, those users must use the same protocols and services to exchange email messages, and those protocols must be defined as Internet standards.

TCP/IP's Architecture

A key part of understanding the distributed nature of TCP/IP is the realization that TCP/IP is a modular *family* of protocols, providing a wide range of highly segmented functions. TCP/IP is not a single monolithic protocol, but instead is a collection of protocols that range from application-specific functions like web browsing down to the low-level networking protocols like IP and TCP.

One common tool used for comparing different kinds of protocols is the OSI* Reference Model, which is a simplistic breakdown of networking functions from the physical wiring up to the applications that run on the network. By comparing TCP/IP to the OSI Reference Model, it is easier to understand how each of the major protocols interact with each other.

An Introduction to the OSI Reference Model

The OSI Reference Model is a conceptual model that uses seven "layers" to identify the various functions provided by a network, and these seven layers can be used to compare different protocols using a common framework. Each layer within the OSI Reference Model has a very specific function, and each layer depends on the other layers in order for the entire model to function properly. Each layer only communicates with the layers immediately above or below it. If there is a problem at one layer, it is the responsibility of that specific layer to provide feedback to the layers surrounding it.

The OSI Reference Model is extremely useful as a tool for discussing various network services. For example, if we were to look at a simple network service such as printing a document to a locally attached printer, we could use the OSI Reference Model to determine how this simple task was being achieved. We could also use the model to determine how printing over a Novell network was done, or how printing over a TCP/IP network was accomplished. Because all three of these examples use the same model, they can all be compared to each other even though they all use extremely different technologies to achieve the same objective.

Not all networking technologies have seven layers, nor do they all match up to the seven layers in the OSI Reference Model exactly. Most of them do not match it except in small, specific ways, although all of them can be compared to the model with a little bit of thought. This flexibility is what makes it such a popular tool.

The following list briefly describes each of the seven layers and the purpose each serve. Remember that this is a conceptual model, with very little direct meaning to the real world.

The physical layer

The physical layer is concerned with the physical wiring used to connect different systems together on the network. Examples include serial and parallel cables, Ethernet and Token Ring cabling, telephone wiring, and even the specific connectors and jacks used by these cabling systems. Without strictly standardized definitions for the cabling and connectors, vendors might not implement them in such a way that they would function with other implementations, which in turn would make it impossible for any communication

* OSI stands for Open Systems Interconnect, an alternate suite of network protocols.

whatsoever to occur. Each of these wiring systems therefore follows very strict standards, ensuring that network devices will at least be able to communicate without having to worry about issues such as voltage and impedance.

The data-link layer

The data-link layer defines how information is transmitted across the physical layer, and is responsible for making sure that the physical layer is functioning properly. Some networks—such as the public telephone network, radio, and television—use analog sine-waves to transmit information, while most computer networks use square-wave pulses to achieve this objective. If there are any problems with transmitting the information on the physical cabling (perhaps due to a damaged wire or circuit), then this layer must deal with those errors, either attempting to retransmit the information or reporting the failure to the network layer.

The network layer

The network layer is used to identify the addresses of systems on the network, and for the actual transmission of data between the systems. The network layer must be aware of the physical nature of the network, and package the information in such a way that the data-link layer can deliver it to the physical layer. For example, if a telephone line is the physical layer, then the network layer must package the information in such a way that the data-link layer can transmit it over an analog circuit. Likewise, if the physical layer is a digital Ethernet LAN, then the network layer must encapsulate the information into digital signals appropriate for Ethernet, and then pass it to the data-link layer for transmission.

On many networks, the network layer does not provide any integrity checking. It simply provides the packaging and delivery services, assuming that if the data-link layer is not reporting any errors then the network is operational. Broadcast television and radio work in this manner, assuming that if they can transmit a signal, then it can also be received. Many digital networking technologies also take this approach, leaving it up the higher-level protocols to provide delivery tracking and reliability guarantees.

The transport layer

The transport layer provides the reliability services lacking from the network layer, although only for basic transmission services, and not for any application- or service-specific functions. The transport layer is responsible for verifying that the network layer is operating efficiently, and if not, then the transport layer either requests a retransmission or returns an error to the layer above it. Since higher-level services have to go through the transport layer, all transport services are guaranteed when this layer is designed into the network software and used. Not all systems mandate that the transport layer provide reliability, and many networks provide unreliable transport layers for nonessential services such as broadcast messages.

The session layer

The session layer is responsible for establishing connections between systems, applications, or users. The session layer may receive this request from any higher layer, and then will negotiate a connection using the lower layers. Once a connection is established, the session layer simply provides an interface to the network for the higher layers to communicate with. Once the higher layers are finished, the session layer is responsible for destroying the connection.

The presentation layer

The presentation layer provides a consistent set of interfaces for applications and services to utilize when establishing connections through the session layer. Although these interfaces could also exist at the session layer, that would burden it unnecessarily. It is better to have the session layer only manage sessions and not worry about verifying data or providing other extended services. An example of a service provided by the presentation layer is data-compression, allowing applications to take advantage of the performance gains that compression provides without forcing the applications to develop these services themselves, and without forcing the transport layer to provide this service when it may not always be needed.

The application layer

Finally, the application layer provides the network's interface to end-user application protocols such as HTTP and POP3. This layer should not be confused with the part of the end-user application that displays data to the end user. That function is an entirely separate service, and is outside the scope of the OSI Reference Model.

Although every network must use all seven layers of the OSI Reference Model in some form or another, not every network design provides distinct protocols or services that match all seven layers precisely. TCP/IP is one such networking design, with many layers that do not match up to each of the layers used by the OSI Reference Model.

Comparing TCP/IP to the OSI Reference Model

TCP/IP does not strictly conform to the OSI Reference Model. Some portions of the OSI Reference Model map directly to some of the protocols and services provided by TCP/IP, while many of the layers do not map to each other directly at all. For example, the actual delivery of data over the network is handled at the physical layer, and in this case, the wire is the physical layer. There are no services in TCP/IP that correspond with the physical or data-link layers. Rather, IP passes data to a network adapter's device driver, which provides an interface to the data-link layer in use with the physical layer.

Figure 1-4 shows how TCP/IP matches up with the OSI Reference Model. Notice that TCP/IP does not provide any physical or data-link layer services directly, but instead relies on the local operating system for those services.

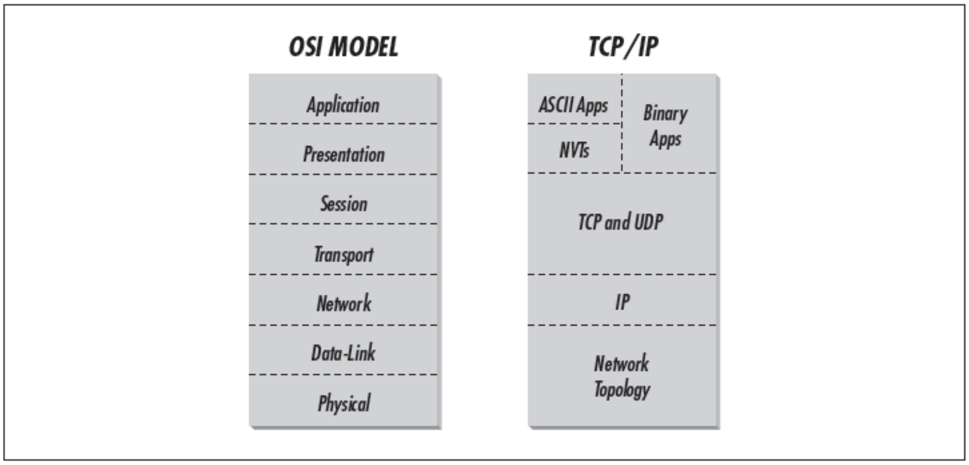


Figure 1-4. TCP/IP in comparison to the OSI Reference Model

The specific layers offered by TCP/IP include:

The Internet Protocol
: IP itself works at the network layer of the OSI reference model. It is responsible for tracking the addresses of devices on the network, determining how IP datagrams are to be delivered, and sending IP packets from one host to another across a specific segment. In essence, IP provides a virtual representation of the network that is independent of any of the individual network segments, acting more like a national delivery service than a local courier service.

The Transport Protocols (TCP and UDP)
: TCP/IP provides two protocols that work at the transport layer: TCP and UDP. TCP provides a highly monitored and reliable transport service, while UDP provides a simple transport with no error-correcting or flow-control services. It is also interesting to note that TCP and UDP also provide session layer services, managing all of the connections between the different hosts. When an application protocol such as HTTP is used to exchange data between a web client and a web server, the actual session-management for this exchange is handled by TCP.

Presentation Services

TCP/IP does not provide a presentation layer service directly. However, some applications use a character-based presentation service called the Network Virtual Terminal (NVTs are a subset of the Telnet specification), while others might use IBM's NetBIOS or Sun's External Data Representation (XDR) programming libraries for this service. In this regard, TCP/IP has many presentation layer services that it *can* use, but it does not have a formal service that every application protocol *must* use.

Application Protocols (HTTP, SMTP, etc.)

TCP/IP provides an assortment of application protocols, providing the end-user applications with access to the data being passed across the transport protocols. These protocols include the Simple Message Transfer Protocol (SMTP), which is used by electronic mail systems to move mail messages around the Internet, and the Hyper-Text Transfer Protocol (HTTP), which is used by web browsers to access data stored on web servers, among many others.

All of these services get called upon whenever an application wants to exchange data with another application across the Internet. For example, a mail client will use the SMTP application protocol whenever a user wants to send a mail message to a remote mail server, and the SMTP protocol uses rules defined by the NVT specification whenever it exchanges data with TCP. In turn, TCP provides error-correction and flow-control services back to SMTP. IP is used to move the TCP segments between the source and destination networks, while hardware-specific protocols (like Ethernet-specific framing) will be used to move the IP packets between the various systems on the network itself.

TCP/IP Protocols and Services In-Depth

Whenever data is exchanged between two applications across a TCP/IP network, each of the major layers provided by TCP/IP come into play.

This can be seen with email clients that use the Simple Message Transfer Protocol (SMTP) to send mail to a local server, as is shown in Figure 1-5. The email software on the client contains local application-specific code for parsing and displaying email messages, but everything else is done with network protocols such as SMTP, TCP, and IP.

As data is passed through each of the different layers, packets are generated that contain two distinct elements: headers and data. As information is passed down through the protocol stack, each layer encapsulates the previous layer's information (including both the header and the data) into a new packet, containing a new layer-specific header and the newly minted data segment. This process is shown in Figure 1-6.

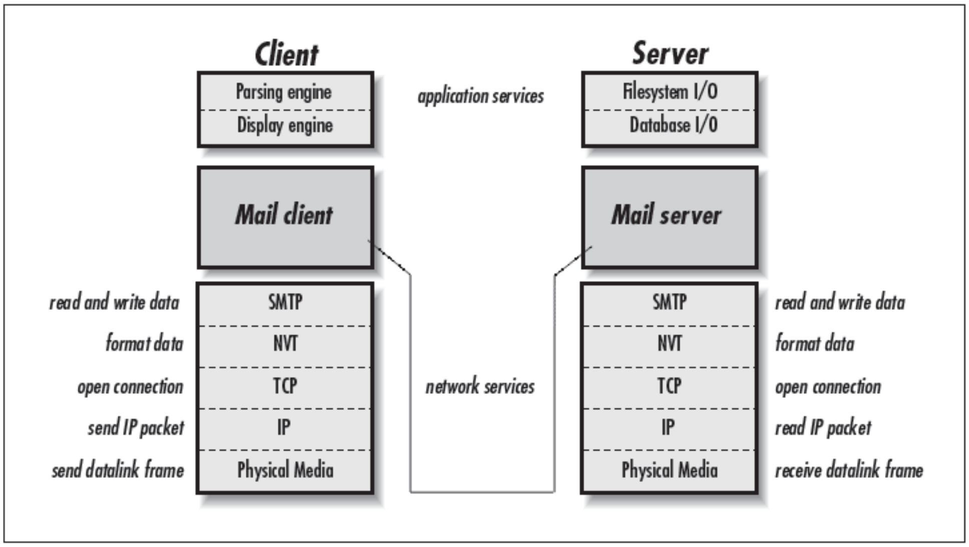


Figure 1-5. Some of the layers used by TCP/IP applications

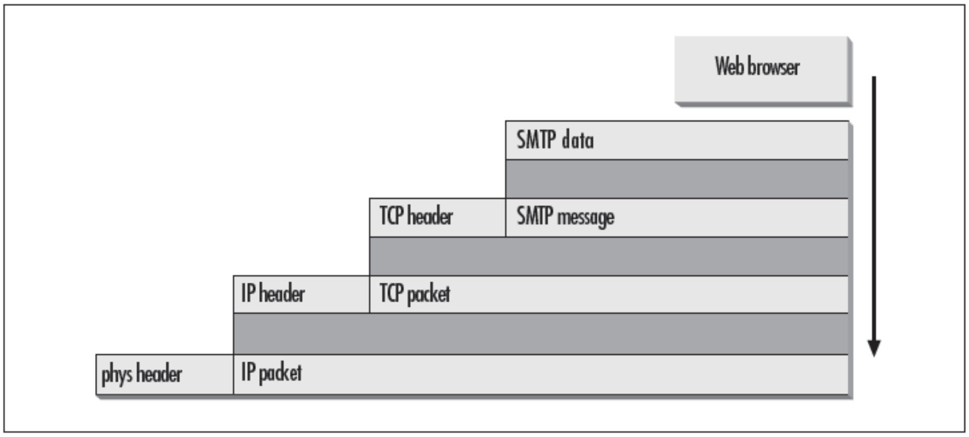


Figure 1-6. The sub-parts of layers

At the bottom-most layer, the physical network is used to transfer bits of data (called "frames") between two devices on the network. IP packets are contained within these network-specific frames. The only reason IP is used for this process is because the data can go over a variety of different network topologies, and as such the TCP/IP applications must have a way of addressing and routing traffic consistently, regardless of the specific networks in use.

Embedded within the IP datagrams are TCP segments, which provide a reliable virtual circuit for the SMTP application protocol to use. TCP does things like open a connection between two application protocol end-points, resend lost data, remove duplicates, and exert flow control, each of which is beyond the simple delivery function of IP itself, yet is common enough to be useful as a separate, distinct service.

The SMTP application protocol contains application-specific semantics. In this case, this might consist of an SMTP command such as "RCPT TO ehall" and an application-specific response code such as 250 ("okay"). Note that the commands and data used by SMTP conform to the NVT specification, which prescribes how the data should be formatted, the types of data allowed, and so forth, although SMTP is doing all of the real work.

As can be seen, each of the layers in the TCP/IP suite provide specific functionality to the layers above and below it, making the overall design extremely modular. It is this modularity that makes TCP/IP so powerful, and also what makes it so complex.

Data-Link Services

When two devices on a network communicate with each other, they don't use IP to do so. Rather, they use protocols that are specific to the wire itself. For example, devices on an Ethernet segment use a predefined series of electrical impulses to communicate with each other. Whenever an Ethernet device wants to send data to another device on the same network, it raises and lowers the voltage of the shared medium so that a series of "on" and "off" voltage patterns are generated. These changes in voltage are interpreted as bits by the other devices on the network.

The changes in voltage are dictated by protocols that are specific to the different types of physical networks. Ethernet networks have data-link protocols that will not work with technologies like Token Ring. Similarly, modems use protocols specific to different types of modem technology.

Much of IP's functionality is determined by the physical media that the IP device is connected to. When an IP device has information that it needs to send to another device on the same wire, it has to understand the characteristics of the wire in order to prepare the information so that is usable for that particular medium.

One of the issues that IP has to deal with is the mechanisms used for the network-specific addressing. Just as physical networks have to provide mechanisms for encapsulating and disseminating data on the wire, they also have to provide a way for devices to locate each other, using addressing methods defined by the low-level protocols.

On shared networks, each device must have a unique hardware address in order for devices to indicate which node the traffic is for. Ethernet networks use a 48-bit

Media Access Control (MAC) address for this purpose, while Frame Relay networks use Data-Link Connection Identifier (DLCI) addresses, and so on. This concept is illustrated in Figure 1-7, where IP traffic for 192.168.10.40 is sent to the Ethernet address of 00:00:c0:c8:b3:27, using Ethernet-specific signalling.

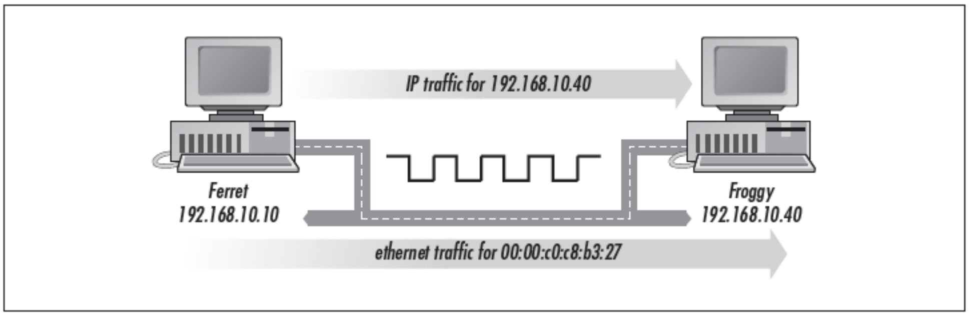

IP traffic for 192.168.10.40
Ferret
192.168.10.10
Froggy
192.168.10.40
ethernet traffic for 00:00:c0:c8:b3:27

Figure 1-7. Topology-specific protocols and addressing

In contrast, modems are point-to-point; only two devices can communicate over any given circuit. As such, modem circuits don't use addresses per se, but instead just send and receive data over dedicated "transmit" and "receive" wires as needed. The same is true of T-1 lines and most other point-to-point circuit-based networks.

In all of these cases, the IP stack running on the local device must understand the addressing mechanisms used by the hardware, and implement it accordingly, just as it must understand the framing characteristics and signalling mechanisms in use on the physical network.

The Internet Protocol

When an IP-enabled device wants to send data to another IP node, the data-link services on that device convert the IP datagrams into a format usable by the local network medium, and then send the data to the destination system using the addressing and framing mechanisms dictated by the network.

These steps occur on each of the networks that an IP datagram traverses on its way to the final destination system. If an IP datagram were sent from a dial-up user working at her home in Los Angeles to a server in Rome, Italy, the number of networks that would be crossed could be quite high. But at each step of the way, the data would be transmitted using the low-level protocols appropriate for each of the particular networks being crossed.

In this regard, IP provides a virtual representation of the global Internet to the hosts that are on it. IP provides a datagram formatting and addressing mechanism that is not dependent upon any of the specific characteristics of the individual networks that make up the global Internet. Data can be sent to an IP address, and the data will be encapsulated and transmitted according to the rules of each of the intermediary networks, with the IP datagram being used to provide delivery clues to the sending, receiving, and intermediary devices. Essentially, routing occurs at the network layer (IP), while delivery occurs at the data-link layer (Ethernet, modems, whatever).

This concept is illustrated in Figure 1-8. In that example, data sent over a modem would be encapsulated into a form usable by the dial-up connection. Once received, the data would be determined to be an IP datagram, and would then get converted into a form that was usable by the LAN connection and sent out again. The receiving system (Ferret) would eventually get the packets.

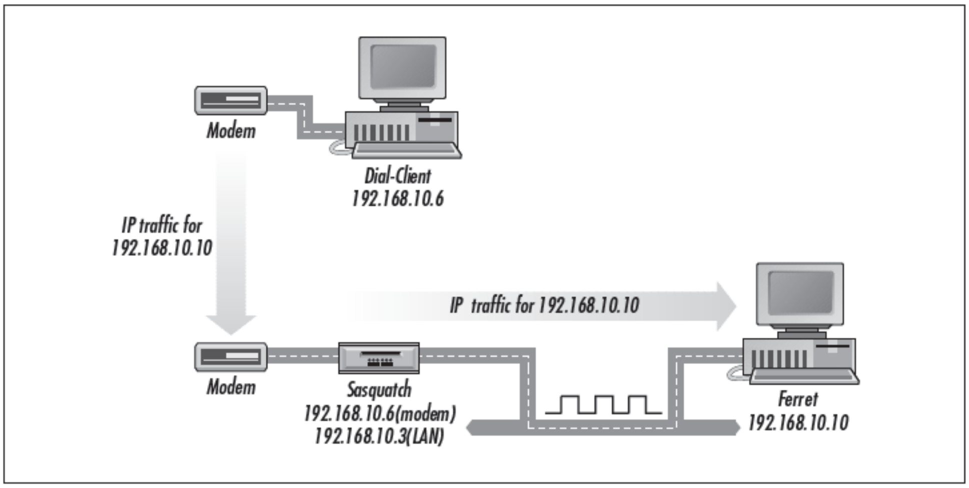


Figure 1-8. IP datagrams versus the topology-specific protocols

One thing to keep in mind is that this was the primary design goal of IP, allowing it to scale beyond the packet-switched networks that made up the original Internet (which could not be grown easily or cheaply). Without moving to a virtual networking protocol like IP, the Internet would still be using packet-switching networks, and we'd all have WAN connections on our desktops instead of Ethernet or Token Ring (or, more likely, we wouldn't be using IP). But by leveraging the virtual nature of IP, we can use any network we want anywhere we want, and the IP data will still be deliverable across any of them.

One side effect of this design is that the IP datagram is a separate entity from the IP packets that are being used to transfer the datagram from the source to the destination. Whenever a device needs to send data, it will form an IP datagram containing both the data that needs to be sent and whatever headers are required to deliver the data over IP to the destination system. However, as this datagram is sent across the network, it will be shipped as a series of packets that get created and destroyed by each network device that processes or forwards the datagram on to its final destination. In essence, the datagram becomes a series of packets, each of which can go anywhere they need to in order for the datagram to be delivered.

Another interesting aspect of IP is that it does not guarantee that any of these packets will ever get delivered at all. A system may be able to send the data, but the data may not be received intact, or the data may be ignored by the destination system due to high processing loads or some other reason. Although some networking topologies provide an intelligent retransmission mechanism in case data is lost, many of them do not. As such, IP's designers had to assume that data would get lost sometimes.

In this regard, IP offers absolutely no guarantees, leaving it up to higher-layer protocols to perform this function if required. For this reason, IP can be thought of as being unreliable, particularly in the sense that application designers (and users) should not assume that every IP datagram will arrive at its destination intact. Some people refer to this as "best-effort" delivery, while others refer to it jokingly as "best-of-luck" delivery.

Another key design goal of IP was the concept of datagram independence. The IP protocol does not dictate that all datagrams must travel the same route. In fact, IP dictates just the opposite: any datagram can travel across any network path that the devices on the network deem most suitable.

For example, a user in California may be downloading data from a host in Ireland, and some part of the network may simply stop functioning. The sending system (or a router somewhere in between the two systems) would eventually detect this failure and would begin forwarding datagrams through a different network path. This feature gives IP a robust and flexible foundation, allowing networks to become self-healing, in a sense.

Since each datagram is independent, it is likely that some datagrams will take different paths to the same destination. As such, one datagram may end up crossing a satellite link, while another datagram crosses a fiber-optic line. When this happens, the second datagram will likely arrive at the destination system before the first datagram does. In another situation, the satellite link may experience some sort of problem that results in the first datagram getting sent twice.

In both of these cases, the network has caused the IP datagrams to get out of synch. But since IP is simply a virtual representation of the network, it does not care when this happens. If sequencing is important to an application, then it has to implement that service directly or by using TCP (appropriately the Transmission Control Protocol) for transport-layer services.

Another related concept is fragmentation. Assume for a moment that the sending system were on a high-capacity network such as Token Ring, while the destination system were on a low-capacity dial-up connection. Since the sending system generates datagrams according to the characteristics of the local network, it generates large datagrams appropriate for the Token Ring frames.

But when the next system tries to relay the IP datagram to the dial-up recipient, the IP datagrams are too large for the dial-up network to handle in one frame. When this happens, the datagram must be split into multiple fragments, with each of the fragments being sent across the network as independent entities. Each fragment follows all of the other rules outlined earlier, thus being capable of getting lost in transit, routed separately, or arriving out of sequence.

When the fragments arrive at the destination system (*if* they arrive), then they need to be reassembled into their original datagram form. However, this only occurs at the final destination system, and not at any intermediary routers, since any of the fragments could have gone off in another direction.

Taken together, these services make IP a highly unreliable and unpredictable network protocol. Datagrams can get lost or may be broken into multiple packets, all without warning. The only thing IP does is move data from one host to another, one network at a time. Of course, users have little interest in applications that must be provided by a higher-level protocol than either IP itself (for example, TCP) or the application.

Remember this rule: The Internet Protocol is *only* responsible for getting IP datagrams from one host to another, one network at a time.

For more information on IP, refer to Chapter 2, *The Internet Protocol*.

The Address Resolution Protocol

Since two IP devices on the same physical medium communicate with each other using the low-level protocols specific to that physical medium, the two devices must locate each other's hardware addresses before they can exchange any data. However, each networking topology has its own addressing mechanisms that are different from all of the others, and IP has to be able to locate hardware addresses for each of them.

Since there are so many different types of networking topologies, it is not possible for IP to be imbued with the specific knowledge of how to build the address mappings for each of them explicitly. Attempting to do so would be an extraordinarily inefficient use of the IP software's basic functionality, preventing the rapid adoption of new networking topologies and introducing other fundamental problems into the network.

Instead, the Address Resolution Protocol (ARP) is used as a helper to IP, and is called upon to perform the specific task of building each address mapping whenever address conversion is required. ARP works by issuing a broadcast on the selected medium, requesting that the device using a specific IP address respond with its hardware address. Once the destination device has responded, the sending system can establish communication with the receiving system and start sending data to the discovered hardware address. This process is shown in Figure 1-9, with 192.168.10.10 issuing a lookup for 192.168.10.40, who responds with its local Ethernet hardware address.

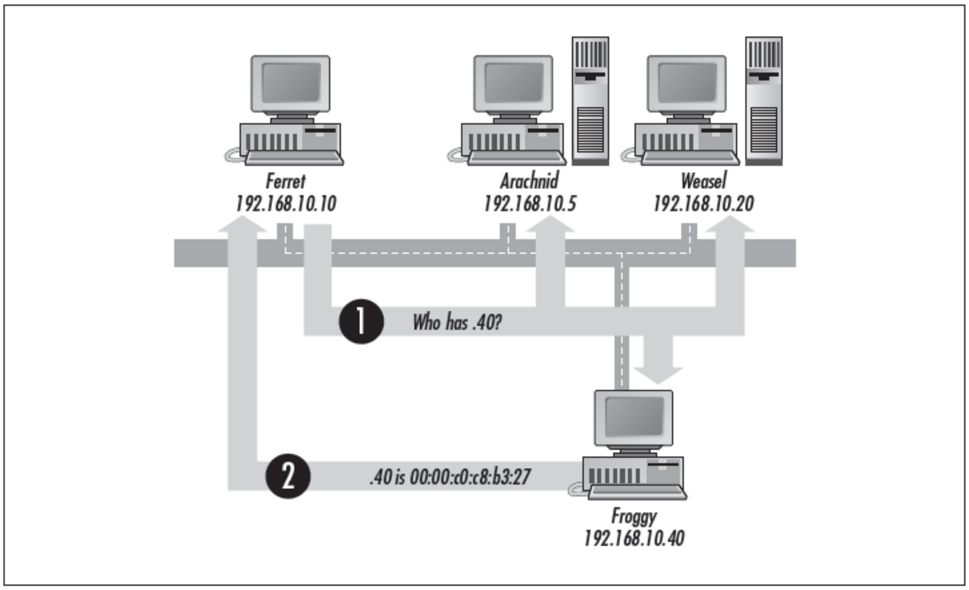


Figure 1-9. Using ARP to locate the hardware address associated with a known IP address

The ARP requests and responses work at the physical layer and are embedded directly into the frames provided by the low-level protocols in use on the physical medium. ARP messages do not use IP, but instead are entirely separate protocols.

For more information on ARP, refer ahead to Chapter 3, *The Address Resolution Protocol*.

The Internet Control Message Protocol

From time to time, IP datagrams will fail to get delivered. This may be due to errors in the datagram structure, a general network outage, or a delivery timeout. IP doesn't really care about these problems, since it never promised delivery in the first place. However, applications care about these problems very much. They would like to be able to react to a failure, either by taking an alternative course of action, or by at least informing the user that a problem has occurred.

IP uses the Internet Control Message Protocol (ICMP) for error-reporting services. When a system needs to report a problem that is preventing delivery from occurring, it generates an ICMP message that describes the general problem, and then sends it back to the original sender of the original datagram. ICMP messages are not sent when a packet is lost in transit or when some other transient error occurs. Rather, ICMP error messages are only sent when there is a detectable problem that is preventing certain packets or datagrams from being delivered due to a specific reason. This indicates that the sending host should probably stop trying to send those kinds of datagrams to this specific destination system, or that a different path should be used.

Even if two IP-enabled systems are able to communicate effectively, there are no guarantees that everything will work, since the data inside the datagrams may be corrupt, or packets may get lost without any ICMP errors being generated. IP is an unreliable network protocol by its very definition, and as such does not provide any guarantees. ICMP does not change this fact.

ICMP runs on top of IP, allowing it to traverse the global Internet just as easily as TCP or UDP messages. This seems a bit confusing to many people: if an IP datagram could not be delivered, it would not seem that an ICMP error—delivered over IP—would make it back to the original sender. However, remember that most delivery errors occur due to problems on the *next* leg of the network, and that the original IP datagram at least made it as far as the system that's reporting a problem. In this scenario, the network between the original sender and the host that's reporting the problem is likely to be functioning properly.

There are a variety of ICMP message types, and not all of them are limited to reporting on network errors. There are also ICMP "query" messages, useful for diagnosing and testing the network interactively. The most common of these are the ICMP Echo Request and Echo Reply query messages, which are better known as *ping* to most users.

For more information on ICMP, refer to Chapter 5, *The Internet Control Message Protocol.*

The Transport Protocols

Application protocols do not communicate with IP directly, but instead talk to one of two transport protocols: TCP or UDP. In turn, these transport protocols pass data to IP, which encapsulates the data into IP datagrams that get sent over the network. In essence, the transport protocols hide the network from the application protocols so that they do not have to deal with packet-sizing and other issues, while also shielding the network from having to multiplex the application protocol traffic (a task that IP can leave to the transport protocols).

For example, both UDP and TCP provide a multiplexing service to application protocols by way of application-specific port numbers. Essentially, port numbers act as virtual post office boxes for messages to be delivered to within a single host, allowing multiple applications to run on a single host. When datagrams arrive at a destination system, they are handed off to the transport protocol specified in the datagram, which then delivers the transport-specific message data to the port number specified in the header of the message. In this manner, many different application protocols can run on the same host, using different port numbers to identify themselves to the transport protocols.

The transport protocol that an application protocol uses is determined by the kinds of network- and application-management services that are required. TCP is a reliable, connection-oriented transport protocol, providing error-correction and flow-control services that can tolerate IP's knack for losing packets. Conversely, UDP is an unreliable, message-centric transport protocol that offers little functionality over IP alone. There are many applications that need to use one of these models or the other, and there are a handful of applications that use both of them. A good example of an application that could use them both is a network printer.

If many users share a network printer, all of them need to be kept informed of the printer's availability. Using UDP, the printer could send out periodic status messages such as "out of paper" or "cover open." The software on the client PCs would then pick up on these status updates, changing the desktop icon appropriately, or notifying an administrator that something has gone awry. UDP allows the printer to notify everyone of these updates simultaneously, since it's not a big deal if some of these updates get lost.

This concept is illustrated in Figure 1-10, in which the printer at 192.168.10.2 is periodically sending out UDP broadcasts, indicating its current status. Network systems that are interested in that information can monitor for those updates and can change their desktop icons or management station software appropriately. If a system does not receive one of these updates for some reason, then it will probably get the next update message, so it's not a big deal.

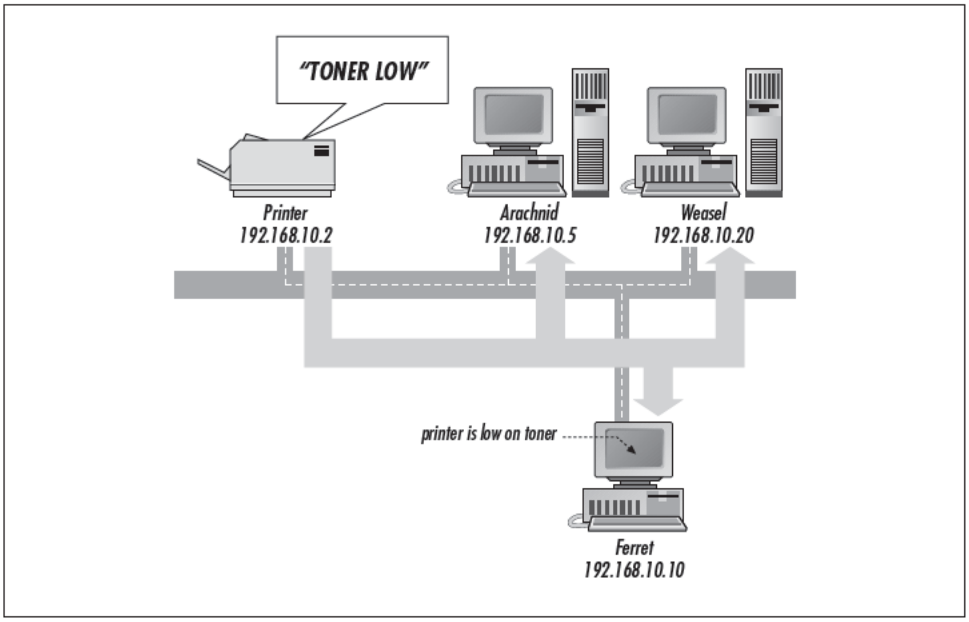


Figure 1-10. Using UDP to broadcast status updates

Conversely, when a user wants to print a file, she would prefer to use TCP, since that would ensure that the printer received all of the data intact. When the user wants to print, the client software on the end user's PC establishes a TCP session with the printer, sends the data to the printer's software, and then closes the connection once the job was submitted.

If the printer is functioning properly, it accepts the data, and uses the error-correction and flow-control services offered by TCP to manage the data transfer. If the printer is not available (perhaps it is out of paper, or low on toner), then it sends an appropriate message using the existing TCP connection. This ensures that the client is notified of whatever problem is preventing the print job from being serviced.

This process is illustrated in Figure 1-11. Here, the desktop PC is trying to print a file to the printer, but since the printer is out of toner, it rejects the connection. Because TCP is a reliable, circuit-centric protocol, the client is sure to get the message, even if it didn't get all of the UDP broadcasts sent earlier.

As you can see, both TCP and UDP provide functionality that is above that offered by IP alone, and both protocols are required to build an effective set of network applications.

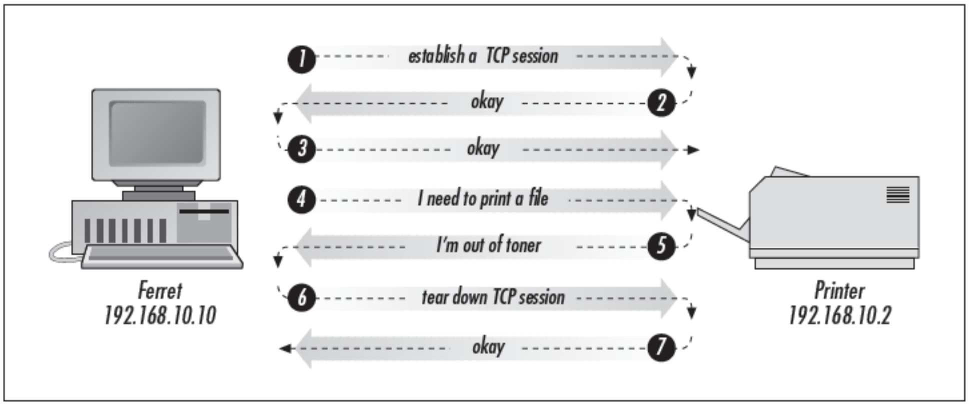


Figure 1-11. Using TCP for transaction-oriented applications

The Transmission Control Protocol

TCP provides error-correction through the use of a connection-oriented transaction. Whenever an application needs to send data to another host, TCP builds a "start" segment and sends it to the destination node. When the other system sends a "start" segment back (along with an acknowledgment that it got the first segment), a monitored conversation between the two systems begins.

TCP works in much the same way as a telephone conversation. When an application wants to trade data with another system, it first tries to establish a workable session. This is similar to you calling another person on the phone. When the other party answers ("Hello?"), they are acknowledging that the call went through. You then acknowledge the other party's acknowledgment ("Hi Joe, this is Eric"), and begin exchanging information.

If at any time during the call parts of the data exchange are lost ("Sorry, what did you say?"), the sender retransmits the questionable data. If the connection degrades to a point where no communication is possible, then sooner or later both parties simply stop talking. Otherwise, once all of the data has been exchanged, the parties agree to disconnect ("See ya"), and close the call gracefully. TCP follows most of these same rules, as is illustrated in Figure 1-12.

TCP segments are encapsulated within IP datagrams. They still rely on IP to get the data where it's going. However, since IP doesn't offer any guarantees regarding delivery, TCP has to keep track of the status of the connection at all times. This is achieved through the use of sequence numbers and acknowledgment flags embedded within the TCP header. Every byte of data sent over TCP must be acknowledged (although these acknowledgments are usually clumped together). If one of the systems does not acknowledge a segment, then TCP will resend the

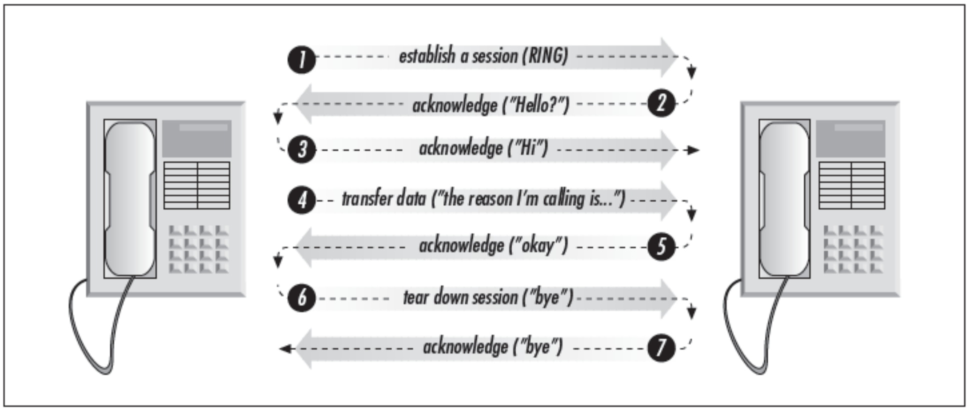

1 establish a session (RING)
acknowledge ("Hello?") 2
3 acknowledge ("Hi")
4 transfer data ("the reason I'm calling is...")
acknowledge ("okay") 5
6 tear down session ("bye")
acknowledge ("bye") 7

Figure 1-12. TCP virtual circuits versus telephone calls

questionable data. This provides error correction and recovery functions that overcome IP's unreliable nature.

The use of sequence numbers also allows TCP to implement flow control and other services on top of IP. Applications can send as much data as they need to, and TCP will break the data into chunks that will fit within IP segments. If the receiving system is unable to process data quickly enough, it can tell the sending system to slow down, thereby reducing the likelihood that data will get lost.

In addition, it is important to realize that TCP offers a byte-stream service for applications to use whenever they need to read and write data. Whenever an application needs to send data—whether that data is a 20-byte message or a two-megabyte file—the application can send the data in a stream to TCP, where it will be converted into manageable chunks of data that are sent (and tracked) over IP cleanly. Once the IP datagrams are received by the destination system, the data is made available to the destination application immediately, where it can be read and processed.

Applications such as the Internet's Simple Message Transport Protocol (SMTP) and Hypertext Transfer Protocol (HTTP) both require the reliable and controlled connection services that TCP provides. In addition, these types of application protocols also benefit from TCP's streaming model, allowing the applications to send data as a continual stream of bytes that will be read and processed by the recipient upon their arrival. Without these services, mail messages sent over SMTP and GIF images sent over HTTP would not flow smoothly, and would likely get garbled. And since TCP provides these services directly, applications do not have to embed these routines within their internal application code.

For more information on TCP, see Chapter 7, *The Transmission Control Protocol.*

The User Datagram Protocol

Not every application requires guaranteed delivery, and these applications typically use UDP for transport services. Unlike TCP, UDP sends only the data it has received from the application, and makes no pretense towards guaranteed delivery or flow control or anything else. As such, UDP is much like IP, but is the protocol that applications use to communicate with each other, rather than using IP directly.

UDP is much like a postcard. If you were travelling around a foreign country, you might send postcards to friends and family from the different cities that you visit, informing them of recent events. You wouldn't worry about the postcards getting delivered quickly, or even if they got lost entirely, since you'll probably send more postcards from the next town anyway. You wouldn't necessarily *want* the postcards to get lost, but at the same time you wouldn't rely on the postcards for any urgent business (like "send money to the embassy"). For anything important, you'd use the telephone (TCP) to ensure that your message arrived intact and was processed correctly.

You may wonder why a UDP protocol exists, when it would seem that IP could serve the same function. The reason is simple: IP doesn't do anything but get datagrams from one host to another. IP doesn't provide any application interfaces or management services. UDP does provide these services, and it provides a consistent environment for developers to use when writing low-overhead network applications. UDP also provides application multiplexing services through the use of port numbers, allowing many application protocols to be used on a single host. Trying to do this with IP would require either a lot more transport protocols, or an application multiplexing layer within IP directly, neither of which would be very efficient.

Another benefit of UDP is that it offers a message-centric delivery model, allowing chunks of data to be sent as single IP datagrams (instead of being streamed over virtual circuits like they would with TCP). For example, a UDP-based application protocol can write a four-kilobyte block of data to UDP, and that block will be handed to IP directly. IP will then create an IP datagram that contains the entire four kilobytes, and send this data as a series of IP packets to the destination system (according to the rules defined for the network medium in use). Once all of the data arrives, the IP datagram is reassembled and the entire four-kilobyte UDP message will be handed to UDP for processing.

In this model, it is easy for applications to exchange record-oriented data (such as a fixed-length file or a database record), since the entire record can be read by a single operation. Since the IP datagram (and thus the UDP message) will be contained in a single message, if the client has received any of the data, then they will

ROKU EXH. 1002

receive *all* of the data in that message. Conversely, TCP would require that the client continually read the queue, waiting for all of the data to arrive, and having no clear indication of when all the data for that record had arrived (without also using application-specific markers in the data stream, anyway).

Also, applications that need fast turnaround or that already have their own internal error-correction routines can make good use of UDP because of its low overhead. Some database software packages can be configured to use UDP, and many file transfer protocols also use UDP because it is a light, fast, and message-centric protocol that is easier and faster than TCP, and that does not require the overhead of TCP's virtual-circuit model.

For more information on UDP, refer to Chapter 6, *The User Datagram Protocol.*

Presentation Services

Whenever application protocols wish to communicate with each other, they must do so using a predefined set of rules that define the types of data that will be exchanged. For example, if an application protocol is to use textual data, then those characters must have the same byte-order and binary values on both systems. For example, one system cannot use US-ASCII while the other system uses EBCDIC characters. Nor can one system pass data in "big-endian" form to a processor that only understands "little-endian" data, since the bits will be interpreted backwards.

For these reasons, the application protocols must agree to use certain types of data, and must also agree on how to present that data so that it is interpreted consistently. Typically, this falls under the heading of "presentation layer services," with some network architectures providing detailed presentation-layer specifications that cover everything from character sets to numeric formatting rules. However, TCP/IP does not have a formally defined presentation layer. Instead, it has many informal mechanisms that act as presentation layers, with each of them providing specific kinds of presentation services to different kinds of applications.

Most of the application protocols used on the Internet today use the Network Virtual Terminal (NVT) specification for presentation services. NVTs are a subset of the Telnet specification, and provide a basic terminal-to-terminal session that applications use to exchange textual data. The NVT specification defines a simple definition for the characters to use (seven-bit, printable characters from the US-ASCII character set) and end-of-line markers.

However, NVTs do not provide for much in the way of complex data types, such as numeric formatting. If an application needs to exchange a complex piece of data—including extended characters, long integers, and record markers—then NVTs can not be used alone. For this reason, a variety of other presentation-layer services are also used with TCP/IP applications, although typically these services are restricted to vendor-specific applications and offerings.

One presentation service that is popular with Microsoft-based applications is IBM's NetBIOS, a set of network APIs that provide functionality suitable for PC-based network applications. Another popular service is Sun's External Data Representation (XDR) service, a set of APIs that are useful for passing complex data types. Yet another popular service is the Distributed Computing Environment's Remote Procedure Call (DCE RPC) mechanism, useful for passing network-specific data between highly dissimilar hosts.

Each of these mechanisms is popular with different groups and for different reasons. But most Internet-based applications use just NVTs since they are usable on a wide variety of systems. Remember that many of the computing systems in use on the Internet are still quite old and are incapable of supporting anything other than seven-bit ASCII text.

Application Protocols

A variety of application protocols exist that provide standardized mechanisms for the exchange of information across vendor bounds. Among these are file transfer protocols such as FTP, Gopher, and HTTP; groupware and electronic mail services such as SMTP, POP3, IMAP4, and NNTP; and protocols for locating network resources such as DNS, Finger, and LDAP, among many others.

It's important to realize that client applications generally consist of two distinct components: the application protocol (such as HTTP or POP3), and an end-user interface that displays information. For example, a web browser uses HTTP (the protocol) to retrieve HTML and GIFs from a web server, but the code for displaying that data is a separate service that is not covered by the protocol specification.

For more on the common application protocols found on the Internet today, refer to the book *Internet Application Protocols*, which covers most of these protocols.

How Application Protocols Communicate Over IP

Almost all IP applications follow the same basic model: a client sends a request of some kind to a server running on another system, and the server examines the request, acts upon it in some form, and then possibly returns some form of data back to the client. This is not always the case (many UDP-based "servers" do not return any data, but simply monitor network activity), but it holds true for most applications.

Server-based applications (like an email server or web server) are generally loaded by the operating system when the computer is started. The servers then go into a "listen" state, watching for incoming connections. Conversely, client applications will only establish a connection when some sort of action is required (like "get new messages").

Applications communicate with the transport protocols through the use of "ports," which are unique I/O identifiers used by the transport protocols and the specific instance of the application protocol. "Ports" are conceptually similar to the mailboxes used at your local post office. When a letter comes in for a recipient, it is placed into a known mailbox reserved for that specific recipient. Whenever the recipient comes by, he will pick up any messages in that mailbox and process the data at his convenience.

Similarly, ports provide TCP and UDP with a way to deliver data to higher-layer application protocols. Every time an application protocol opens a connection to one of the transport protocols, it will allocate a port from the transport protocol, and then use that port for all network I/O. Any traffic that is destined for that particular application will be routed to the appropriate port for the application to deal with.

Just as every device on an IP network has a unique IP address, every instance of every application protocol also has a unique port number that is used to identify it to the transport protocols on the local system. This concept is illustrated in Figure 1-13, which shows how UDP reserves ports for specific applications. Any UDP or TCP messages that come into a system will be identified as destined for a specific port number, and the transport layer will use that information to route the data to the correct application.

Some applications can open many simultaneous network connections, and in this case, each instance would get its own port number. One example of this is the ubiquitous web browser, which can open many simultaneous connections to a remote web server, depending on the number of files that need to be downloaded from a web page. Each of these HTTP connections will get created as independent network connections, with each of the connections having unique port numbers for the client side of the connection. Once the web browser finishes downloading the objects, then each of the individual connections will be closed.

Every connection between a client and a server consists of four pieces of information: a source IP address, a source port number, a destination address, and a destination port number. All together, these four pieces of information make connections unique. For example, if a web browser were to open two connections to a web server, then the IP addresses of both hosts would be the same. In addition, the well-known server port number (80) would also be the same.

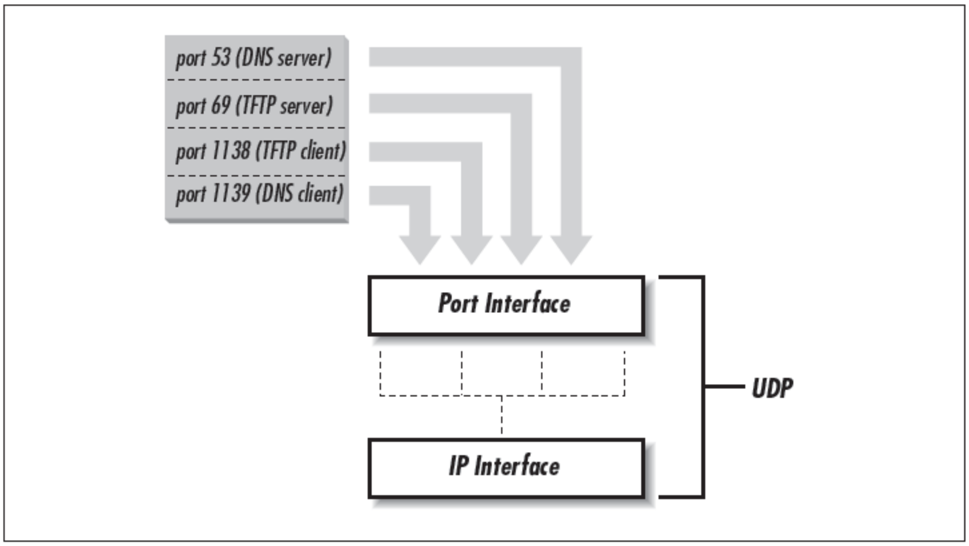


Figure 1-13. Application-level multiplexing with port numbers

Therefore, in order for the individual connections to be unique, the client must use a different port number for each of the unique connections. Servers do not care if a single client asks for multiple connections, as long as each connection comes from a unique port number on the client, since each connection must be uniquely identifiable to the server.

This four-way identifier is called a "socket pair" in IP lingo, and is the basis of all communications for all application protocols. A "port" identifies a connection point in the local stack (i.e., port number 80). A "socket" identifies an IP address and port number together (i.e., port 80 on host 192.168.10.20 could be written as "socket 192.168.10.20:80."). A "socket pair" refers to a distinct connection between two different applications, including the IP addresses and port numbers in use by both. Each individual connection requires that the socket pair contain at least one unique element.

Servers Listen for Incoming Connections

Most server-based IP applications use what are referred to as "well-known" port numbers. For example, an HTTP server will listen on TCP port 80 by default, which is the well-known port number for an HTTP server. This way, any HTTP client that connects to HTTP servers can use the default of TCP port 80. Otherwise, the client would have to specify the port number of the server that it wanted to connect with (you've seen this in some URLs that use *http://www.somehost.com:8080/* or the like, where "*8080*" is the port number of the HTTP server on *www.somehost.com*).

ROKU EXH. 1002

Most application servers allow you to use any port number you want. However, if you were to run your web server on TCP port 8080 for example, then you would have to tell every Internet user that your web server was *not* accessible on TCP port 80. This would be an impossible task. By sticking with the default, all users can connect to your web server using the default of TCP port 80.

Some network administrators purposefully run application servers on non-standard ports, hoping to add an extra layer of security to their network. However, it is the author's opinion that security through obscurity is no security at all, and this method should not be relied upon by itself.

There are a number of predefined port numbers that are registered with the Internet Assigned Numbers Authority (IANA). All of the port numbers below 1024 are reserved for use with well-known applications, although there are also many applications that use port numbers outside of this range.

In addition to the reserved addresses that are managed by the IANA, there are also "unreserved" port numbers that can be used by any application for any purpose, although conflicts may occur with other users who are also using those port numbers. Any port number that is frequently used is encouraged to register with the IANA.

For a detailed listing of all of the port numbers that are currently registered, refer to the IANA's online registry (accessible at *http://www.isi.edu/in-notes/iana/assignments/port-numbers*). To see the well-known ports used on your system, examine the */etc/services* file on a Unix host, or the *C:\WinNT\System32\Drivers\Etc\SERVICES* file on a Windows NT host.

Clients Open Connections to Servers

In contrast to server-based applications that are always listening for incoming connections on a fixed port number, client applications will use a randomly assigned port number for their end of the connection. Whenever an IP application needs to send data, the transport protocol will allocate a random port number above 1024 and use this port number for all incoming and outgoing data associated with that application.

For example, when a POP3 client is used to establish a connection with a mail server, the client application will pass an application-specific command to TCP, specifying the server's IP address and port number as the destination. TCP will then add its own information—including stuff like the port number of the local

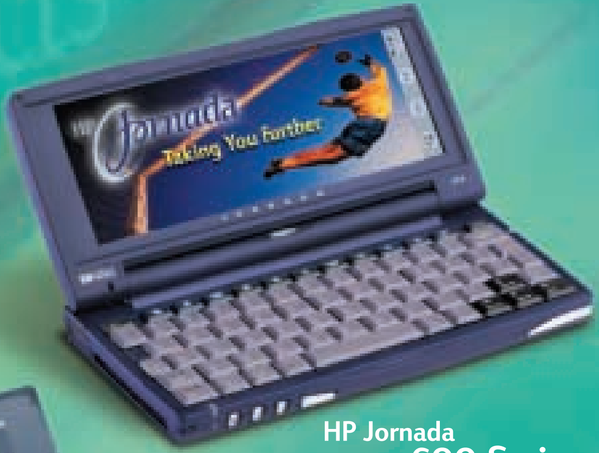
POP3 client—and hand the entire package off to IP for delivery. IP then does its best to get the message to the destination system.

When the mail server's IP stack receives the datagram, it verifies that it contains a TCP segment, and then hands the contents off to TCP for further processing. TCP will see that the destination port number refers to the local POP3 server, and then hand off the original application command. Once the server has processed the command, it will reverse the process, sending whatever data it generates back to the port number and IP address in use by the client. Once the transaction is finished, the client's TCP port will be released. Any subsequent connections would require a new connection be opened, with a different port number being allocated by the client.

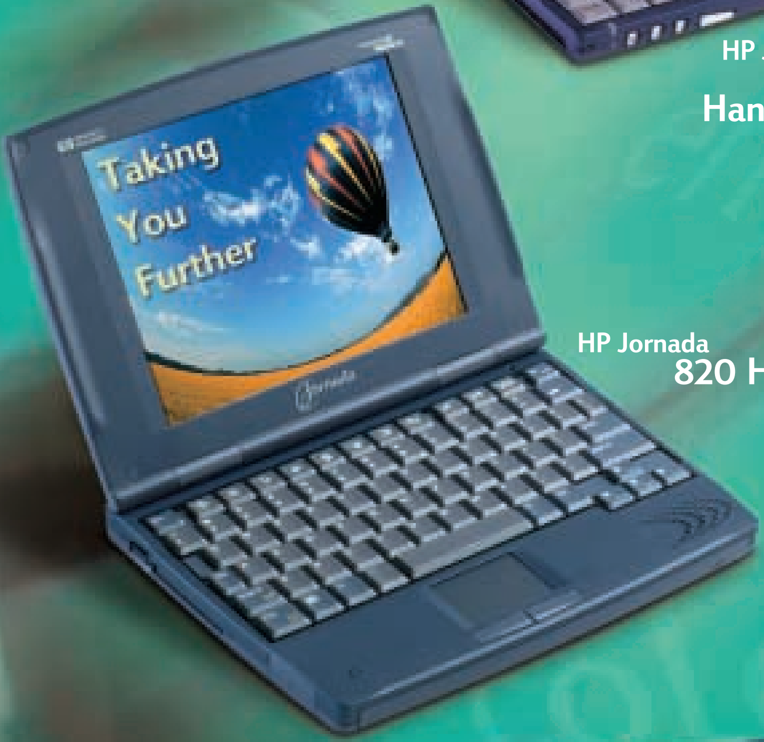
APPENDIX N

HEWLETT® PACKARD

Expanding Possibilities



HP Jornada
600 Series
Handheld PCs



HP Jornada
820 Handheld PC

HP | Jornada | PC Companion Products

HP Jornada
430 Series
Palm-size PCs

HP Jornada

ROKU EXH. 1002

HP Jornada

HEWLETT® PACKARD
Expanding Possibilities

HP Jornada 430 Series Palm-size PCs

Part Number
- HP F1796A #001—HP Jornada 430
- HP F1796A #ABA—HP Jornada 430se

Processor
- 133MHz 32-bit Hitachi SH7709a processor

Memory
- 16MB RAM
- 8MB ROM

Display
- 240 × 320 pixels LCD
- 16-bit; 65,536 colors (selectable)
- CSTN backlit

Input
- Pen-and-touch interface (stylus included)
- CIC Jot Pro
- On-screen keyboard
- 4 user-configurable quick launch screen icons
- 4 quick keys (Record, Exit, Start, and Scroll/Action)—can be disabled

Power
- One standard Lithium-Ion battery
- 7 hours[1] of battery life
- AC adapter

Dimensions
- 5.1 × 3.2 × 0.9 in (13 × 8.1 × 2.2 cm)

Weight
- 8.8 oz (250 g) with standard battery

Ports/Slots
- One IrDA infrared port
- One RS232 serial port
- One CompactFlash Type I and II card slot

Printer Ready
- N/A

Operating System
- Microsoft® Windows® CE 2.11

Personal Information Management
- HP easy contacts[2]
- HP voice contacts[2, 3]
- Microsoft Pocket Outlook (Calendar, Tasks, Contacts, Inbox)
- Microsoft Outlook 2000 (full retail desktop version)[4]

Presentation/Word Processing
- Note Taker
- PhatWare™ HPC Notes™ 3.0 Lite, HPC Notes 2.03, and HPC Spell[2, 3]

Financial Management
- LandWare OmniSolve™[2]

Communications
- Microsoft Channels browser
- AvantGo AvantGo.com™[2, 3]
- Utopiasoft™ Hum™ 1.61[2]
- Audible AudiblePlayer™ 1.0 for Windows CE and AudibleManager™ 2.0[2, 3]
- Trio® PhoneManager 2.0[2, 3]

Connectivity
- Microsoft ActiveSync®

Other Software
- EZOS® EzExplorer™[2]
- Voice Recorder
- BSQUARE bTASK™[2]
- Sierra Imaging Image Expert® CE 2.0[2, 3]
- HP Jornada settings
- HP Jornada backup
- Inso® Quick View Plus®[2, 6]
- World Clock
- Solitaire
- Sampler songs from *Chang and Eng—The Musical* and *Echoboys*
- Sample skins for Hum MP3 player

Warranty
- 1 year std. (optional extended warranty available)

HP Jornada 600 Series Handheld PCs

Part Number
- HP F1262A—HP Jornada 680 (with internal modem)
- HP F1263A—HP Jornada 680e (without internal modem)
- HP F1813A—HP Jornada 690 (with internal modem)
- HP F1814A—HP Jornada 690e (without internal modem)

Processor
- 133MHz 32-bit Hitachi SH3 processor

Memory
- 16MB RAM; upgradable (HP Jornada 680)
- 32MB RAM (HP Jornada 690)
- Burst Mode ROM; upgradable

Display
- 6.5-in (16.7-cm) LCD; 65,536 colors
- 640 × 240 pixels on screen
- 1024 × 768 pixels on external monitor

Input
- Large (76% full-size) keyboard
- Stylus and touchscreen

Power
- Lithium-Ion rechargeable battery
- One 3V CR2032 coin-cell backup battery
- Up to 8 hours[1] of battery life
- AC adapter

Dimensions
- 7.4 × 3.7 × 1.3 in (18.9 × 9.5 × 3.4 cm)

Weight
- 1.1 lbs (510 g) with standard battery

Ports/Slots
- Serial port (RS-232C)
- Fast IrDA-compliant port
- Modem port (RJ11)
- One PC Card Type II card slot
- One CompactFlash Type I card slot
- Audio speaker and microphone

Printer Ready
- Direct printing via IrDA or serial port to PCL printers

Operating System
- Microsoft Windows CE, Handheld PC Professional Edition

Personal Information Management
- Microsoft Pocket Outlook (Calendar, Tasks, Contacts, Inbox)
- Microsoft Outlook 2000 (full retail desktop version)[4]
- HP Jornada viewer
- HP quick pad

Presentation/Word Processing
- Microsoft Pocket Word
- Microsoft Pocket PowerPoint
- HP Jornada show

Financial Management
- Microsoft Pocket Excel
- Microsoft Pocket Access
- LandWare OmniSolve[5]
- On The Go Software® Pocket Quicken®[7]

Communications
- Internal 56Kbps fax modem[8]
- Microsoft Pocket Internet Explorer
- HP Jornada dialup
- BSQUARE bFAX® Pro[5]

Connectivity
- Microsoft ActiveSync
- Starfish™ TrueSync™ CE 2.0[2]

Other Software
- HP Jornada settings, backup, hot keys, macro, power, country selector
- Microsoft PowerPoint Viewer/Converter
- Inso Outside In®[5]
- BSQUARE bFIND™[5]
- Trio PhoneManager 2.0[5]
- Sierra Imaging Image Expert CE 2.0[2]
- WESTTEK™ *JETCET*™ PRINT HP Lite and *JETCET* Print 2.0 trial copy[2]

Warranty
- 1 year std. (optional extended warranty available)

HP Jornada 820 Handheld PC

Part Number
- HP F1260A—HP Jornada 820 (with internal modem)
- HP F1261A—HP Jornada 820e (without internal modem)

Processor
- 190MHz Intel StrongARM™ RISC processor

Memory
- 16MB RAM
- 16MB Burst Mode ROM; upgradable

Display
- 8.2-in (21.0-cm) VGA CSTN screen
- 640 × 480 pixels on screen
- 1024 × 768 pixels on external monitor

Input
- Touch-typeable (90% full-size) keyboard
- Integrated touch pad

Power
- Lithium-Ion rechargeable battery
- Two 3V CR2032 coin-cell backup batteries
- Up to 10 hours[1] of battery life
- AC adapter

Dimensions
- 9.7 × 7.0 × 1.3 in (24.6 × 17.8 × 3.3 cm)

Weight
- 2.5 lbs (1.1 kg) with standard battery

Ports/Slots
- Serial port (RS-232C)
- Fast IrDA-compliant port
- Monitor (DB15) and modem (RJ11) ports
- Universal Serial Bus (USB) host support
- One PC Card Type II card slot
- One CompactFlash Type II card slot
- Audio speaker and microphone

Printer Ready
- Direct printing via IrDA or serial port to PCL printers

Operating System
- Microsoft Windows CE, Handheld PC Professional Edition

Personal Information Management
- Microsoft Pocket Outlook (Calendar, Tasks, Contacts, Inbox)
- HP Jornada viewer

Presentation/Word Processing
- Microsoft Pocket Word
- Microsoft Pocket PowerPoint
- HP Jornada show

Financial Management
- Microsoft Pocket Excel
- Microsoft Pocket Access
- LandWare OmniSolve[5]

Communications
- Internal 56Kbps fax modem[8]
- Microsoft Pocket Internet Explorer
- HP Jornada dialup
- BSQUARE bFAX Pro[5]

Connectivity
- Microsoft ActiveSync
- Starfish TrueSync CE 2.0[2]

Other Software
- HP Jornada hot keys, settings, backup
- Inso Outside In[5]
- BSQUARE bFIND[5]
- Trio PhoneManager 2.0[5]
- Sierra Imaging Image Expert CE 2.0[2]
- WESTTEK *JETCET* PRINT HP Lite and *JETCET* Print 2.0 trial copy[2]

Warranty
- 1 year std. (optional extended warranty available)

[1] Estimated battery life. Actual battery life will vary based on usage.
[2] Available in English only.
[3] Not included with the HP Jornada 430 Palm-size PC.
[4] Microsoft Outlook 2000 is not available in Portuguese or Chinese.
[5] Available in English for North America, Europe, Latin America, and Asia Pacific only. Trio PhoneManager is downloadable from the Web for U.S. users.
[6] Not included with the HP Jornada 430se Palm-size PC.
[7] Available for U.S. only.
[8] Model with internal modem may not be available in some countries. Please check with your reseller for details.

Powered by Microsoft® Windows® CE Palm-size PC

Powered by Microsoft® Windows® CE Handheld PC

Printed in the U.S.A. 11/99

To learn more about HP's Jornada family of PC companion products, please visit us on the Web at www.hp.com/jornada

APPENDIX O

RangeLAN2™ 7410 CE PC Card

Mobilizing the Workforce with Handheld PCs

Data Sheet

7410

Advantages

- **Priced to Mobilize Handheld PC Devices**
  Low-priced RangeLAN2 wireless LAN adapter, exclusively for use in Windows™ CE Handheld PCs.
- **Industry's Lowest Power Consumption**
  Marathon™ power management delivers less than 5 mA current draw in doze mode, significantly extending battery life.
- **Driver Built Into Windows CE Operating System**
  Driver built into Windows CE 2.11 (Professional Edition 3.0) for plug-and-play utilization. Standard driver available for Windows CE 2.0.
- **Fast**
  Delivers a 1.6 Mbps data rate for responsive network connectivity.

Increase Your Productivity—Take Your Network with You

The RangeLAN2 7410 CE PC Card is a high performance wireless LAN adapter designed to meet the needs of mobile users who require continuous LAN connectivity. Delivering an optimal combination of range, throughput, and low power consumption, it is the perfect solution for users of Windows™ CE Handheld devices. And best of all, it is value priced to deliver cost effective mobile computing.

Imagine being able to take your handheld anywhere in your workplace with unbroken network connectivity. Scan your email while listening to a lecture. Call up an important file during a meeting. Use the web to verify a crucial fact. With the RangeLAN2 7410 CE PC Card and a RangeLAN2 or OpenAir™ network backbone in place in your facility, all this is not just possible, but today it is reality for thousands of satisfied customers worldwide.

Power Management That Keeps You On the Move

With Marathon™ Power Management, the RangeLAN2 7410 CE PC Card keeps you mobile longer. When your device is neither transmitting nor receiving but needs to be network aware, the RangeLAN2 7410 slips into doze mode, reducing the current draw to below 5 mA, significantly increasing battery life. Further, the RangeLAN2 7410 features the industry's lowest transmit (265 mA) and receive (130 mA) power consumption.

Guaranteed Interoperability through OpenAir Certification

With the RangeLAN2 7410 CE PC Card, small Windows CE handheld devices can benefit from wireless network connectivity. And since the RangeLAN2 7410 fully complies with the OpenAir™ standard, connectivity and interoperability is guaranteed with all of the large base of OpenAir-certified products from more than twenty companies belonging to the Wireless LAN Interoperability Forum.

Standard Drivers and Software Tools for Easy Network Installation

Proxim's unique Site Survey software is provided with each unit to assist in fast and easy wireless network design and installation. And with drivers built into the Windows CE Professional Edition 3.0 (Windows CE 2.11) Operating System, simple plug-and-play installation gets you up and running quickly. Plus network management has never been easier with Proxim's web-based RangeLAN2 Manager. Using RangeLAN2 Manager software, network managers can install, configure, monitor and troubleshoot their wireless LAN from anywhere on the network.

proxim

ROKU EXH. 1002

RangeLAN2™ 7410 CE PC Card

Specifications

General

Radio Data Rate	1.6 Mbps per channel, 800 Kbps fallback rate
Range (snap-on)	~400 feet (~122 m) radius indoors 700 feet (~213 m) radius outdoors (more with optional dipole antenna)
Channels	Supports up to 15 independent, non-interfering virtual channels (hopping patterns)
Power Management	265 mA transmit, 130 mA receive, less than 5 mA doze mode, 2 mA sleep mode (all are typical values)
Certifications	• US – FCC Part 15 • International – Contact Proxim for a list of currently certified countries
Compatibility	Fully interoperable with all RangeLAN2 or OpenAir™-certified (Wireless LAN Interoperability Forum) products
Warranty	1 year parts and labor (return to factory)

Network Information

Network Architecture	Supports ad hoc peer-to-peer networks and infrastructure communication to wired Ethernet or Token Ring networks via Access and Extension Point(s)
Drivers	Built into Windows™ CE Professional Edition 3.0 (Windows CE OS 2.11); Windows CE 2.0 drivers available for free download on Proxim's web site
Roaming	Seamless Roaming
Domains	Up to 16 domains for simultaneous independent networks
Security	Twenty character alphanumeric encrypted Security ID
Installation & Diagnostics	Site survey tool included. Surveys other wireless units, reports link quality and ping statistics to APs. Desktop icon continuously reports connection status

Radio

Frequency Band	2.4 GHz frequency band. Actual frequencies in use vary by country
Radio Type	Frequency Hopping Spread-Spectrum (FHSS)
Output Power	100 mW
Voltage	5 V
Antenna Options	Standard snap-on (0 dBi gain); Optional dipole (1 dBi gain)

Environmental

Temperature Range	-20 to +60 degrees C (operating) -20 to +65 degrees C (storage)
Humidity (non-condensing)	20 to 90% typical

Physical Specifications

Form Factor	PCMCIA, Type II PC Card. Card and Socket Services 2.1 compliant
Weight	1.09 ounces (31 g), PC Card only

Ordering Information

7410	RangeLAN2 CE PC Card
7741	Optional dipole antenna

Accessories

Snap-on antenna, operating manual

proxim

Proxim, Inc

Corporate Headquarters
510 DeGuigne Drive
Sunnyvale, CA 94086 USA
Phone: 408-731-2700 / 800-229-1630
Fax: 408-731-3675
sales@proxim.com
www.proxim.com

European Headquarters
4, avenue Morane Saulnier
78140 Velizy, France
Phone: +33 (0)1 30 70 61 18
Fax: +33 (0)1 30 70 61 19
europe@proxim.com
www.proxim.com

OpenAir 2.4

APPENDIX P

Specification Volume 1

Specification of the Bluetooth System

Wireless connections made easy

Core

Bluetooth™

v1.0 B
December 1st 1999

ROKU EXH. 1002

BLUETOOTH DOC	Date / Day-Month-Year 01 Dec 99	N.B.	Document No. **1.C.47/1.0 B**
Responsible	e-mail address		Status

Bluetooth™

Specification of the Bluetooth System

Version 1.0 B

Bluetooth

Revision History

The Revision History is shown in Appendix I on page 868

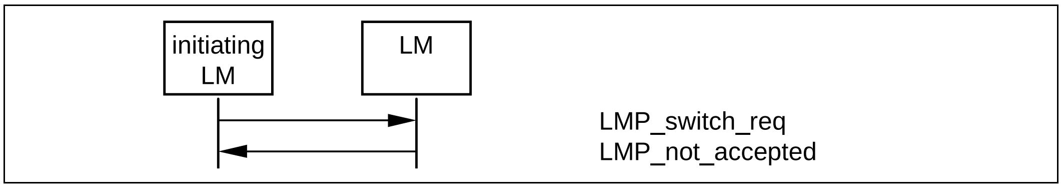
Contributors

The persons who contributed to this specification are listed in
Appendix II on page 879.

Web Site

This specification can also be found on the Bluetooth website:
http://www.bluetooth.com

Disclaimer and copyright notice



Sequence 24: Master-slave switch not accepted.

3.13 NAME REQUEST

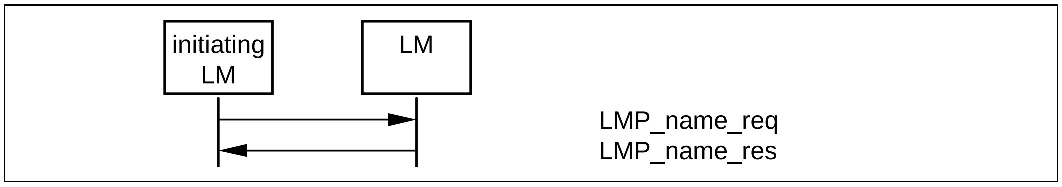
LMP supports name request to another Bluetooth device. The name is a user-friendly name associated with the Bluetooth device and consists of a maximum of 248 bytes coded according to the UTF-8 standard. The name is fragmented over one or more DM1 packets. When the LMP_name_req is sent, a name offset indicates which fragment is expected. The corresponding LMP_name_res carries the same name offset, the name length indicating the total number of bytes in the name of the Bluetooth device and the name fragment, where:

- name fragment(N) = name(N + name offset), if (N + name offset) < name length
- name fragment(N) = 0 ,otherwise.

Here $0 \le N \le 13$. In the first sent LMP_name_req, name offset=0. Sequence 25 is then repeated until the initiator has collected all fragments of the name.

M/O	PDU	Contents
M	LMP_name_req	name offset
M	LMP_name_res	name offset name length name fragment

Table 3.13: PDUs used for name request.



Sequence 25: Device's name requested and it responses.

3.14 DETACH

The connection between two Bluetooth devices can be closed anytime by the master or the slave. A reason parameter is included in the message to inform the other party of why the connection is closed.

M/O	PDU	Contents
M	LMP_detach	reason

Table 3.14: PDU used for detach.

Part E

SERVICE DISCOVERY PROTOCOL (SDP)

This specification defines a protocol for locating services provided by or available through a Bluetooth device.

Bluetooth™

CONTENTS

1 Introduction 327
1.1 General Description 327
1.2 Motivation...327
1.3 Requirements...327
1.4 Non-requirements and Deferred Requirements.......................328
1.5 Conventions 329
1.5.1 Bit And Byte Ordering Conventions...........................329

2 Overview 330
2.1 SDP Client-Server Interaction 330
2.2 Service Record..332
2.3 Service Attribute..334
2.4 Attribute ID 335
2.5 Attribute Value..335
2.6 Service Class 336
2.6.1 A Printer Service Class Example 336
2.7 Searching for Services..337
2.7.1 UUID...337
2.7.2 Service Search Patterns..338
2.8 Browsing for Services 338
2.8.1 Example Service Browsing Hierarchy 339

3 Data Representation 341
3.1 Data Element 341
3.2 Data Element Type Descriptor 341
3.3 Data Element Size Descriptor..342
3.4 Data Element Examples..343

4 Protocol Description..344
4.1 Transfer Byte Order 344
4.2 Protocol Data Unit Format...344
4.3 Partial Responses and Continuation State 346
4.4 Error Handling...346
4.4.1 SDP_ErrorResponse PDU 347
4.5 ServiceSearch Transaction 348
4.5.1 SDP_ServiceSearchRequest PDU.............................348
4.5.2 SDP_ServiceSearchResponse PDU..........................349
4.6 ServiceAttribute Transaction 351
4.6.1 SDP_ServiceAttributeRequest PDU...........................351
4.6.2 SDP_ServiceAttributeResponse PDU........................352

ROKU EXH. 1002

4.7 ServiceSearchAttribute Transaction 354
4.7.1 SDP_ServiceSearchAttributeRequest PDU 354
4.7.2 SDP_ServiceSearchAttributeResponse PDU 356

5 Service Attribute Definitions.. 358
5.1 Universal Attribute Definitions.. 358
5.1.1 ServiceRecordHandle Attribute.................................. 358
5.1.2 ServiceClassIDList Attribute...................................... 359
5.1.3 ServiceRecordState Attribute.................................... 359
5.1.4 ServiceID Attribute 359
5.1.5 ProtocolDescriptorList Attribute.................................. 360
5.1.6 BrowseGroupList Attribute 361
5.1.7 LanguageBaseAttributeIDList Attribute....................... 361
5.1.8 ServiceInfoTimeToLive Attribute 362
5.1.9 ServiceAvailability Attribute.. 363
5.1.10 BluetoothProfileDescriptorList Attribute 363
5.1.11 DocumentationURL Attribute 364
5.1.12 ClientExecutableURL Attribute................................... 364
5.1.13 IconURL Attribute.. 365
5.1.14 ServiceName Attribute 365
5.1.15 ServiceDescription Attribute...................................... 366
5.1.16 ProviderName Attribute... 366
5.1.17 Reserved Universal Attribute IDs............................... 366
5.2 ServiceDiscoveryServer Service Class Attribute Definitions ... 367
5.2.1 ServiceRecordHandle Attribute.................................. 367
5.2.2 ServiceClassIDList Attribute...................................... 367
5.2.3 VersionNumberList Attribute 367
5.2.4 ServiceDatabaseState Attribute 368
5.2.5 Reserved Attribute IDs.. 368
5.3 BrowseGroupDescriptor Service Class Attribute Definitions ... 369
5.3.1 ServiceClassIDList Attribute...................................... 369
5.3.2 GroupID Attribute 369
5.3.3 Reserved Attribute IDs.. 369

Appendix A – Background Information 370

Appendix B – Example SDP Transactions 371

ROKU EXH. 1002

1 INTRODUCTION

1.1 GENERAL DESCRIPTION

The service discovery protocol (SDP) provides a means for applications to discover which services are available and to determine the characteristics of those available services.

1.2 MOTIVATION

Service Discovery in the Bluetooth environment, where the set of services that are available changes dynamically based on the RF proximity of devices in motion, is qualitatively different from service discovery in traditional network-based environments. The service discovery protocol defined in this specification is intended to address the unique characteristics of the Bluetooth environment. See “Appendix A – Background Information,” on page 370, for further information on this topic.

1.3 REQUIREMENTS

The following capabilities have been identified as requirements for version 1.0 of the Service Discovery Protocol.

1. SDP shall provide the ability for clients to search for needed services based on specific attributes of those services.
2. SDP shall permit services to be discovered based on the class of service.
3. SDP shall enable browsing of services without a priori knowledge of the specific characteristics of those services.
4. SDP shall provide the means for the discovery of new services that become available when devices enter RF proximity with a client device as well as when a new service is made available on a device that is in RF proximity with the client device.
5. SDP shall provide a mechanism for determining when a service becomes unavailable when devices leave RF proximity with a client device as well as when a service is made unavailable on a device that is in RF proximity with the client device.
6. SDP shall provide for services, classes of services, and attributes of services to be uniquely identified.
7. SDP shall allow a client on one device to discover a service on another device without consulting a third device.
8. SDP should be suitable for use on devices of limited complexity.
9. SDP shall provide a mechanism to incrementally discover information about the services provided by a device. This is intended to minimize the quantity

of data that must be exchanged in order to determine that a particular service is not needed by a client.

10. SDP should support the caching of service discovery information by intermediary agents to improve the speed or efficiency of the discovery process.
11. SDP should be transport independent.
12. SDP shall function while using L2CAP as its transport protocol.
13. SDP shall permit the discovery and use of services that provide access to other service discovery protocols.
14. SDP shall support the creation and definition of new services without requiring registration with a central authority.

1.4 NON-REQUIREMENTS AND DEFERRED REQUIREMENTS

The Bluetooth SIG recognizes that the following capabilities are related to service discovery. These items are not addressed in SDP version 1.0. However, some may be addressed in future revisions of the specification.

1. SDP 1.0 does not provide access to services. It only provides access to information about services.
2. SDP 1.0 does not provide brokering of services.
3. SDP 1.0 does not provide for negotiation of service parameters.
4. SDP 1.0 does not provide for billing of service use.
5. SDP 1.0 does not provide the means for a client to control or change the operation of a service.
6. SDP 1.0 does not provide an event notification when services, or information about services, become unavailable.
7. SDP 1.0 does not provide an event notification when attributes of services are modified.
8. This specification does not define an application programming interface for SDP.
9. SDP 1.0 does not provide support for service agent functions such as service aggregation or service registration.

1.5 CONVENTIONS

1.5.1 Bit And Byte Ordering Conventions

When multiple bit fields are contained in a single byte and represented in a drawing in this specification, the more significant (high-order) bits are shown toward the left and less significant (low-order) bits toward the right.

Multiple-byte fields are drawn with the more significant bytes toward the left and the less significant bytes toward the right. Multiple-byte fields are transferred in network byte order. See Section 4.1 Transfer Byte Order on page 344.

2 OVERVIEW

2.1 SDP CLIENT-SERVER INTERACTION

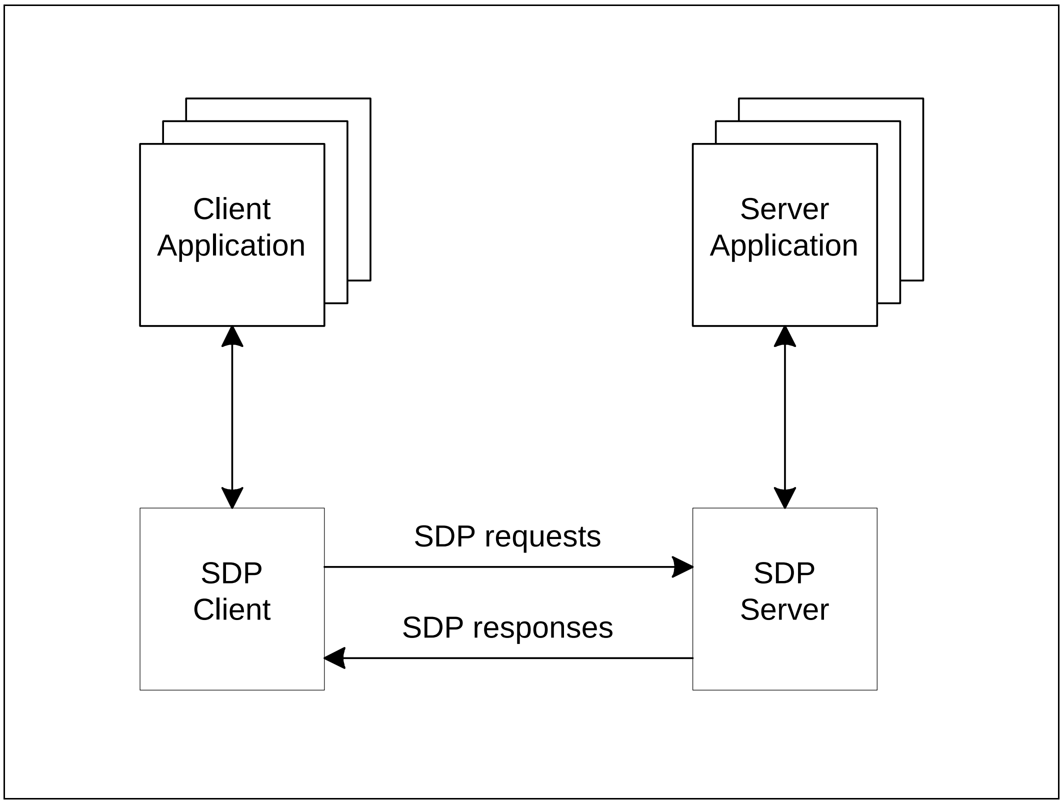


Figure 2.1:

The service discovery mechanism provides the means for client applications to discover the existence of services provided by server applications as well as the attributes of those services. The attributes of a service include the type or class of service offered and the mechanism or protocol information needed to utilize the service.

As far as the Service Discovery Protocol (SDP) is concerned, the configuration shown in Figure 1 may be simplified to that shown in Figure 2.

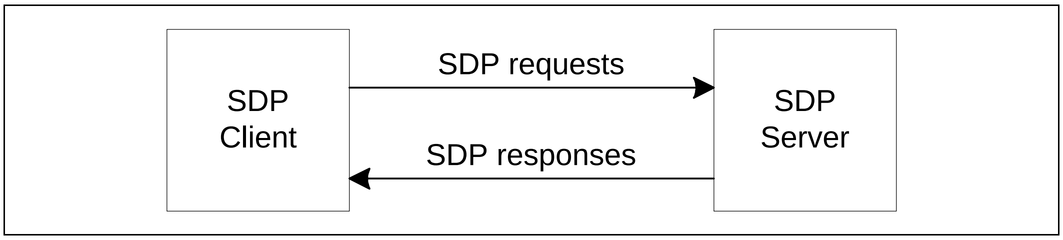


Figure 2.2:

SDP involves communication between an SDP server and an SDP client. The server maintains a list of service records that describe the characteristics of services associated with the server. Each service record contains information about a single service. A client may retrieve information from a service record maintained by the SDP server by issuing an SDP request.

If the client, or an application associated with the client, decides to use a service, it must open a separate connection to the service provider in order to utilize the service. SDP provides a mechanism for discovering services and their attributes (including associated service access protocols), but it does not provide a mechanism for utilizing those services (such as delivering the service access protocols).

There is a maximum of one SDP server per Bluetooth device. (If a Bluetooth device acts only as a client, it needs no SDP server.) A single Bluetooth device may function both as an SDP server and as an SDP client. If multiple applications on a device provide services, an SDP server may act on behalf of those service providers to handle requests for information about the services that they provide.

Similarly, multiple client applications may utilize an SDP client to query servers on behalf of the client applications.

The set of SDP servers that are available to an SDP client can change dynamically based on the RF proximity of the servers to the client. When a server becomes available, a potential client must be notified by a means other than SDP so that the client can use SDP to query the server about its services. Similarly, when a server leaves proximity or becomes unavailable for any reason, there is no explicit notification via the service discovery protocol. However, the client may use SDP to poll the server and may infer that the server is not available if it no longer responds to requests.

Additional information regarding application interaction with SDP is contained in the Bluetooth Service Discovery Profile document.

2.2 SERVICE RECORD

A service is any entity that can provide information, perform an action, or control a resource on behalf of another entity. A service may be implemented as software, hardware, or a combination of hardware and software.

All of the information about a service that is maintained by an SDP server is contained within a single service record. The service record consists entirely of a list of service attributes.

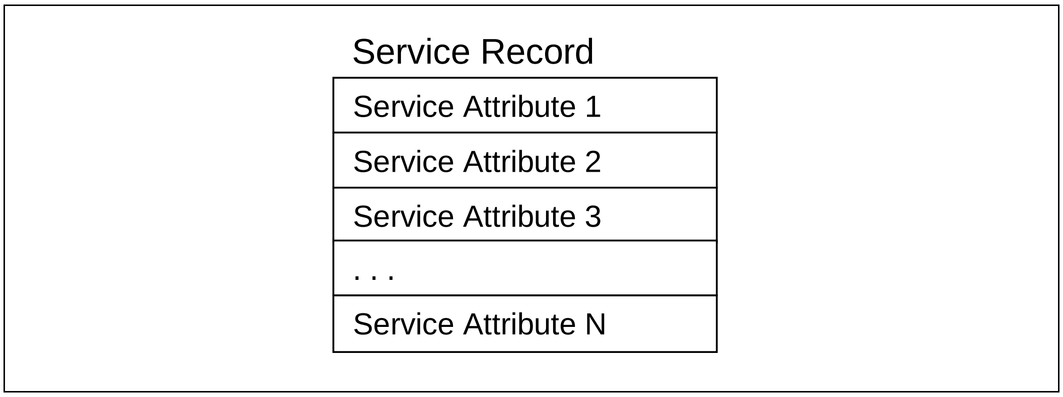


Figure 2.3: Service Record

A service record handle is a 32-bit number that uniquely identifies each service record within an SDP server. It is important to note that, in general, each handle is unique only within each SDP server. If SDP server S1 and SDP server S2 both contain identical service records (representing the same service), the service record handles used to reference these identical service records are completely independent. The handle used to reference the service on S1 will be meaningless if presented to S2.

The service discovery protocol does not provide a mechanism for notifying clients when service records are added to or removed from an SDP server. While an L2CAP (Logical Link Control and Adaptation Protocol) connection is established to a server, a service record handle acquired from the server will remain valid unless the service record it represents is removed. If a service is removed from the server, further requests to the server (during the L2CAP connection in which the service record handle was acquired) using the service's (now stale) record handle will result in an error response indicating an invalid service record handle. An SDP server must ensure that no service record handle values are re-used while an L2CAP connection remains established. Note that service record handles are known to remain valid across successive L2CAP connections while the ServiceDatabaseState attribute value remains unchanged. See the ServiceRecordState and ServiceDatabaseState attributes in Section 5 Service Attribute Definitions on page 358.

There is one service record handle whose meaning is consistent across all SDP servers. This service record handle has the value 0x00000000 and is a handle to the service record that represents the SDP server itself. This service record contains attributes for the SDP server and the protocol it supports. For example, one of its attributes is the list of SDP protocol versions supported by the server. Service record handle values 0x00000001-0x0000FFFF are reserved.

2.3 SERVICE ATTRIBUTE

Each service attribute describes a single characteristic of a service. Some examples of service attributes are:

ServiceClassIDList	Identifies the type of service represented by a service record. In other words, the list of classes of which the service is an instance
ServiceID	Uniquely identifies a specific instance of a service
ProtocolDescriptorList	Specifies the protocol stack(s) that may be used to utilize a service
ProviderName	The textual name of the individual or organization that provides a service
IconURL	Specifies a URL that refers to an icon image that may be used to represent a service
ServiceName	A text string containing a human-readable name for the service
ServiceDescription	A text string describing the service

See Section 5.1 Universal Attribute Definitions on page 358, for attribute definitions that are common to all service records. Service providers can also define their own service attributes.

A service attribute consists of two components: an attribute ID and an attribute value.

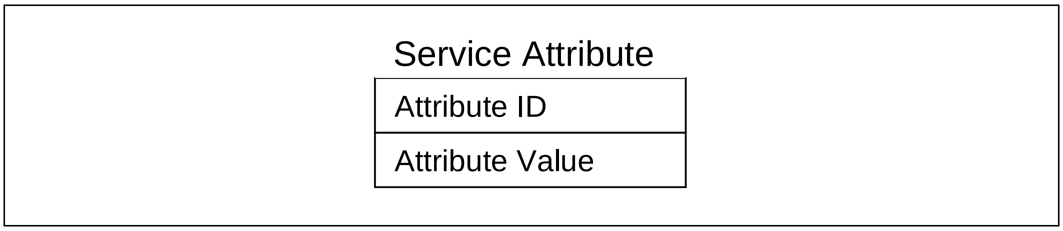


Figure 2.4: Service Attribute

2.4 ATTRIBUTE ID

An attribute ID is a 16-bit unsigned integer that distinguishes each service attribute from other service attributes within a service record. The attribute ID also identifies the semantics of the associated attribute value.

A service class definition specifies each of the attribute IDs for a service class and assigns a meaning to the attribute value associated with each attribute ID.

For example, assume that service class C specifies that the attribute value associated with attribute ID 12345 is a text string containing the date the service was created. Assume further that service A is an instance of service class C. If service A's service record contains a service attribute with an attribute ID of 12345, the attribute value must be a text string containing the date that service A was created. However, services that are not instances of service class C may assign a different meaning to attribute ID 12345.

All services belonging to a given service class assign the same meaning to each particular attribute ID. See Section 2.6 Service Class on page 336.

In the Service Discovery Protocol, an attribute ID is often represented as a data element. See Section 3 Data Representation on page 341.

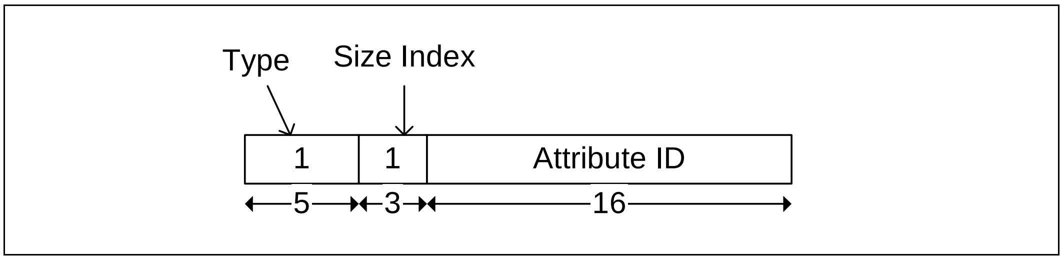


Figure 2.5:

2.5 ATTRIBUTE VALUE

The attribute value is a variable length field whose meaning is determined by the attribute ID associated with it and by the service class of the service record in which the attribute is contained. In the Service Discovery Protocol, an attribute value is represented as a data element. (See Section 3 Data Representation on page 341.) Generally, any type of data element is permitted as an attribute value, subject to the constraints specified in the service class definition that assigns an attribute ID to the attribute and assigns a meaning to the attribute value. See Section 5 Service Attribute Definitions on page 358, for attribute value examples.

2.6 SERVICE CLASS

Each service is an instance of a service class. The service class definition provides the definitions of all attributes contained in service records that represent instances of that class. Each attribute definition specifies the numeric value of the attribute ID, the intended use of the attribute value, and the format of the attribute value. A service record contains attributes that are specific to a service class as well as universal attributes that are common to all services.

Each service class is also assigned a unique identifier. This service class identifier is contained in the attribute value for the ServiceClassIDList attribute, and is represented as a UUID (see Section 2.7.1 UUID on page 337). Since the format and meanings of many attributes in a service record are dependent on the service class of the service record, the ServiceClassIDList attribute is very important. Its value should be examined or verified before any class-specific attributes are used. Since all of the attributes in a service record must conform to all of the service's classes, the service class identifiers contained in the ServiceClassIDList attribute are related. Typically, each service class is a subclass of another class whose identifier is contained in the list. A service subclass definition differs from its superclass in that the subclass contains additional attribute definitions that are specific to the subclass. The service class identifiers in the ServiceClassIDList attribute are listed in order from the most specific class to the most general class.

When a new service class is defined that is a subclass of an existing service class, the new service class retains all of the attributes defined in its superclass. Additional attributes will be defined that are specific to the new service class. In other words, the mechanism for adding new attributes to some of the instances of an existing service class is to create a new service class that is a subclass of the existing service class.

2.6.1 A Printer Service Class Example

A color postscript printer with duplex capability might conform to 4 ServiceClass definitions and have a ServiceClassIDList with UUIDs (See Section 2.7.1 UUID on page 337.) representing the following ServiceClasses:

DuplexColorPostscriptPrinterServiceClassID,
ColorPostscriptPrinterServiceClassID,
PostscriptPrinterServiceClassID,
PrinterServiceClassID

Note that this example is only illustrative. This may not be a practical printer class hierarchy.

2.7 SEARCHING FOR SERVICES

Once an SDP client has a service record handle, it may easily request the values of specific attributes, but how does a client initially acquire a service record handle for the desired service records? The Service Search transaction allows a client to retrieve the service record handles for particular service records based on the values of attributes contained within those service records.

The capability search for service records based on the values of arbitrary attributes is not provided. Rather, the capability is provided to search only for attributes whose values are Universally Unique Identifiers[1] (UUIDs). Important attributes of services that can be used to search for a service are represented as UUIDs.

2.7.1 UUID

A UUID is a universally unique identifier that is guaranteed to be unique across all space and all time. UUIDs can be independently created in a distributed fashion. No central registry of assigned UUIDs is required. A UUID is a 128-bit value.

To reduce the burden of storing and transferring 128-bit UUID values, a range of UUID values has been pre-allocated for assignment to often-used, registered purposes. The first UUID in this pre-allocated range is known as the Bluetooth Base UUID and has the value 00000000-0000-1000-7007-00805F9B34FB, from the Bluetooth Assigned Numbers document. UUID values in the pre-allocated range have aliases that are represented as 16-bit or 32-bit values. These aliases are often called 16-bit and 32-bit UUIDs, but it is important to note that each actually represents a 128-bit UUID value.

The full 128-bit value of a 16-bit or 32-bit UUID may be computed by a simple arithmetic operation.

128_bit_value = 16_bit_value * 2^{96} + Bluetooth_Base_UUID

128_bit_value = 32_bit_value * 2^{96} + Bluetooth_Base_UUID

A 16-bit UUID may be converted to 32-bit UUID format by zero-extending the 16-bit value to 32-bits. An equivalent method is to add the 16-bit UUID value to a zero-valued 32-bit UUID.

Note that two 16-bit UUIDs may be compared directly, as may two 32-bit UUIDs or two 128-bit UUIDs. If two UUIDs of differing sizes are to be compared, the shorter UUID must be converted to the longer UUID format before comparison.

1. The format of UUIDs is defined by the International Organization for Standardization in ISO/IEC 11578:1996. “Information technology – Open Systems Interconnection – Remote Procedure Call (RPC)”

ROKU EXH. 1002

2.7.2 Service Search Patterns

A service search pattern is a list of UUIDs used to locate matching service records. A service search pattern is said to match a service record if each and every UUID in the service search pattern is contained within any of the service record's attribute values. The UUIDs need not be contained within any specific attributes or in any particular order within the service record. The service search pattern matches if the UUIDs it contains constitute a subset of the UUIDs in the service record's attribute values. The only time a service search pattern does not match a service record is if the service search pattern contains at least one UUID that is not contained within the service record's attribute values. Note also that a valid service search pattern must contain at least one UUID.

2.8 BROWSING FOR SERVICES

Normally, a client searches for services based on some desired characteristic(s) (represented by a UUID) of the services. However, there are times when it is desirable to discover which types of services are described by an SDP server's service records without any a priori information about the services. This process of looking for any offered services is termed browsing. In SDP, the mechanism for browsing for services is based on an attribute shared by all service classes. This attribute is called the BrowseGroupList attribute. The value of this attribute contains a list of UUIDs. Each UUID represents a browse group with which a service may be associated for the purpose of browsing.

When a client desires to browse an SDP server's services, it creates a service search pattern containing the UUID that represents the root browse group. All services that may be browsed at the top level are made members of the root browse group by having the root browse group's UUID as a value within the BrowseGroupList attribute.

Normally, if an SDP server has relatively few services, all of its services will be placed in the root browse group. However, the services offered by an SDP server may be organized in a browse group hierarchy, by defining additional browse groups below the root browse group. Each of these additional browse groups is described by a service record with a service class of BrowseGroupDescriptor.

A browse group descriptor service record defines a new browse group by means of its Group ID attribute. In order for a service contained in one of these newly defined browse groups to be browseable, the browse group descriptor service record that defines the new browse group must in turn be browseable. The hierarchy of browseable services that is provided by the use of browse group descriptor service records allows the services contained in an SDP server to be incrementally browsed and is particularly useful when the SDP server contains many service records.

2.8.1 Example Service Browsing Hierarchy

Here is a fictitious service browsing hierarchy that may illuminate the manner in which browse group descriptors are used. Browse group descriptor service records are identified with (G); other service records with (S).

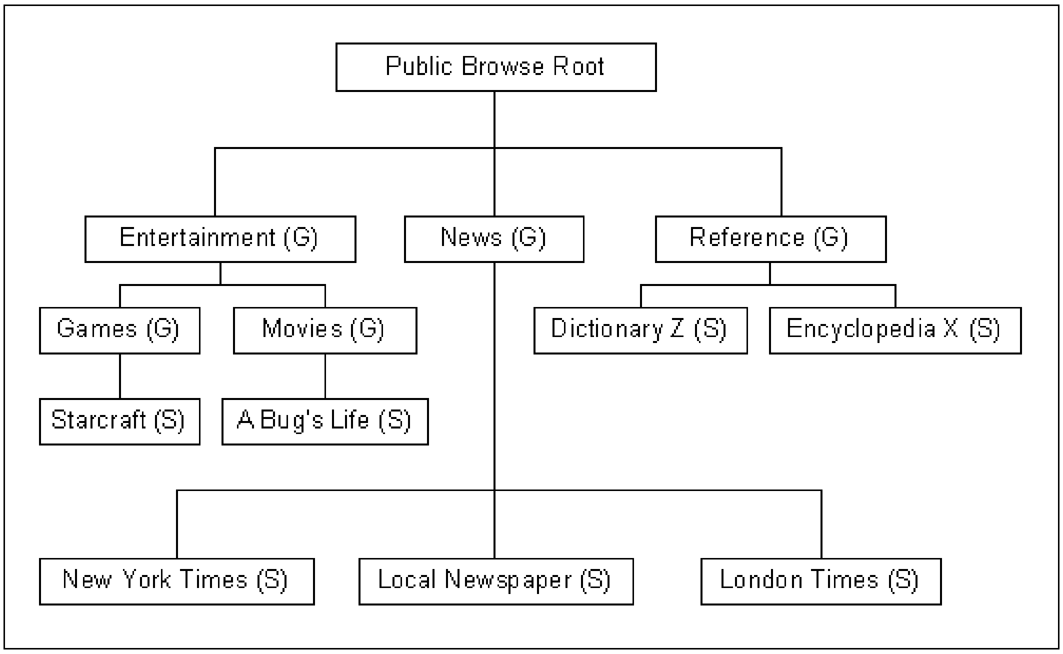


Figure 2.6:

This table shows the services records and service attributes necessary to implement the browse hierarchy.

Service Name	Service Class	Attribute Name	Attribute Value
Entertainment	BrowseGroupDescriptor	BrowseGroupList	PublicBrowseRoot
		GroupID	EntertainmentID
News	BrowsegroupDescriptor	BrowseGroupList	PublicBrowseRoot
		GroupID	NewsID
Reference	BrowseGroupDescriptor	BrowseGroupList	PublicBrowseRoot
		GroupID	ReferenceID
Games	BrowseGroupDescriptor	BrowseGroupList	EntertainmentID
		GroupID	GamesID
Movies	BrowseGroupDescriptor	BrowseGroupList	EntertainmentID
		GroupID	MoviesID
Starcraft	Video Game Class ID	BrowseGroupList	GamesID

Table 2.1:

ROKU EXH. 1002

A Bug’s Life	Movie Class ID	BrowseGroupList	MovieID
Dictionary Z	Dictionary Class ID	BrowseGroupList	ReferenceID
Encyclopedia X	Encyclopedia Class ID	BrowseGroupList	ReferenceID
New York Times	Newspaper ID	BrowseGroupList	NewspaperID
London Times	Newspaper ID	BrowseGroupList	NewspaperID
Local Newspaper	Newspaper ID	BrowseGroupList	NewspaperID

Table 2.1:

3 DATA REPRESENTATION

Attribute values can contain information of various types with arbitrary complexity; thus enabling an attribute list to be generally useful across a wide variety of service classes and environments.

SDP defines a simple mechanism to describe the data contained within an attribute value. The primitive construct used is the data element.

3.1 DATA ELEMENT

A data element is a typed data representation. It consists of two fields: a header field and a data field. The header field, in turn, is composed of two parts: a type descriptor and a size descriptor. The data is a sequence of bytes whose length is specified in the size descriptor (described in Section 3.3 Data Element Size Descriptor on page 342) and whose meaning is (partially) specified by the type descriptor.

3.2 DATA ELEMENT TYPE DESCRIPTOR

A data element type is represented as a 5-bit type descriptor. The type descriptor is contained in the most significant (high-order) 5 bits of the first byte of the data element header. The following types have been defined.

Type Descriptor Value	**Valid Size Descriptor Values**	**Type Description**
0	0	Nil, the null type
1	0, 1, 2, 3, 4	Unsigned Integer
2	0, 1, 2, 3, 4	Signed twos-complement integer
3	1, 2, 4	UUID, a universally unique identifier
4	5, 6, 7	Text string
5	0	Boolean
6	5, 6, 7	Data element sequence, a data element whose data field is a sequence of data elements
7	5, 6, 7	Data element alternative, data element whose data field is a sequence of data elements from which one data element is to be selected.
8	5, 6, 7	URL, a uniform resource locator
9-31		Reserved

Table 3.1:

3.3 DATA ELEMENT SIZE DESCRIPTOR

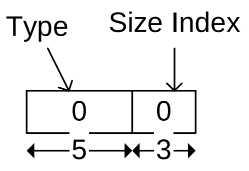
The data element size descriptor is represented as a 3-bit size index followed by 0, 8, 16, or 32 bits. The size index is contained in the least significant (low-order) 3 bits of the first byte of the data element header. The size index is encoded as follows.

Size Index	Additional bits	Data Size
0	0	1 byte. Exception: if the data element type is nil, the data size is 0 bytes.
1	0	2 bytes
2	0	4 bytes
3	0	8 bytes
4	0	16 bytes
5	8	The data size is contained in the additional 8 bits, which are interpreted as an unsigned integer.
6	16	The data size is contained in the additional 16 bits, which are interpreted as an unsigned integer.
7	32	The data size is contained in the additional 32 bits, which are interpreted as an unsigned integer.

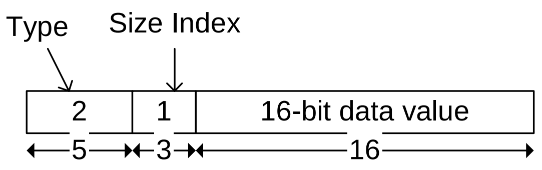
Table 3.2:

3.4 DATA ELEMENT EXAMPLES

Nil is represented as:



A 16-bit signed integer is represented as:



he 3 character ASCII string "Hat" is represented as:

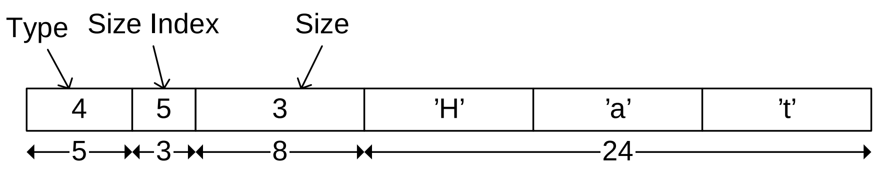


Figure 3.1:

4 PROTOCOL DESCRIPTION

SDP is a simple protocol with minimal requirements on the underlying transport. It can function over a reliable packet transport (or even unreliable, if the client implements timeouts and repeats requests as necessary).

SDP uses a request/response model where each transaction consists of one request protocol data unit (PDU) and one response PDU. However, the requests may potentially be pipelined and responses may potentially be returned out of order.

In the specific case where SDP utilises the Bluetooth L2CAP transport protocol, multiple SDP PDUs may be sent in a single L2CAP packet, but only one L2CAP packet per connection to a given SDP server may be outstanding at a given instant. Limiting SDP to sending one unacknowledged packet provides a simple form of flow control.

The protocol examples found in Appendix B – Example SDP Transactions, may be helpful in understanding the protocol transactions.

4.1 TRANSFER BYTE ORDER

The service discovery protocol transfers multiple-byte fields in standard network byte order (Big Endian), with more significant (high-order) bytes being transferred before less-significant (low-order) bytes.

4.2 PROTOCOL DATA UNIT FORMAT

Every SDP PDU consists of a PDU header followed by PDU-specific parameters. The header contains three fields: a PDU ID, a Transaction ID, and a ParameterLength. Each of these header fields is described here. Parameters may include a continuation state parameter, described below; PDU-specific parameters for each PDU type are described later in separate PDU descriptions.

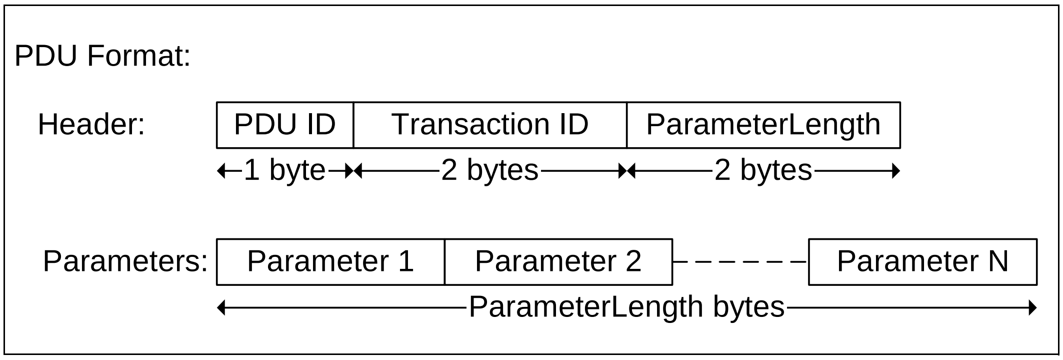


Figure 4.1:

ROKU EXH. 1002

PDU ID: *Size: 1 Byte*

Value	Parameter Description
N	The PDU ID field identifies the type of PDU. I.e. its meaning and the specific parameters.
0x00	Reserved
0x01	SDP_ErrorResponse
0x02	SDP_ServiceSearchRequest
0x03	SDP_ServiceSearchResponse
0x04	SDP_ServiceAttributeRequest
0x05	SDP_ServiceAttributeResponse
0x06	SDP_ServiceSearchAttributeRequest
0x07	SDP_ServiceSearchAttributeResponse
0x07-0xFF	Reserved

TransactionID: *Size: 2 Bytes*

Value	Parameter Description
N	The TransactionID field uniquely identifies request PDUs and is used to match response PDUs to request PDUs. The SDP client can choose any value for a request’s TransactionID provided that it is different from all outstanding requests. The TransactionID value in response PDUs is required to be the same as the request that is being responded to. Range: 0x0000 – 0xFFFF

ParameterLength: *Size: 2 Bytes*

Value	Parameter Description
N	The ParameterLength field specifies the length (in bytes) of all parameters contained in the PDU. Range: 0x0000 – 0xFFFF

4.3 PARTIAL RESPONSES AND CONTINUATION STATE

Some SDP requests may require responses that are larger than can fit in a single response PDU. In this case, the SDP server will generate a partial response along with a continuation state parameter. The continuation state parameter can be supplied by the client in a subsequent request to retrieve the next portion of the complete response. The continuation state parameter is a variable length field whose first byte contains the number of additional bytes of continuation information in the field. The format of the continuation information is not standardized among SDP servers. Each continuation state parameter is meaningful only to the SDP server that generated it.

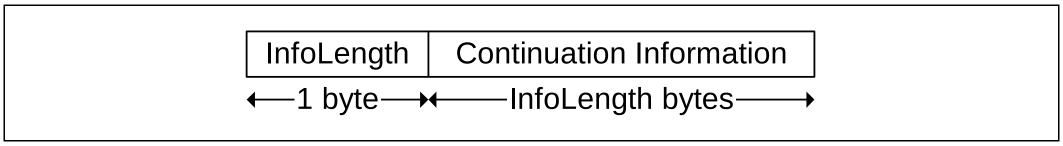


Figure 4.2: Continuation State Format

After a client receives a partial response and the accompanying continuation state parameter, it can re-issue the original request (with a new transaction ID) and include the continuation state in the new request indicating to the server that the remainder of the original response is desired. The maximum allowable value of the InfoLength field is 16 (0x10).

Note that an SDP server can split a response at any arbitrary boundary when it generates a partial response. The SDP server may select the boundary based on the contents of the reply, but is not required to do so.

4.4 ERROR HANDLING

Each transaction consists of a request and a response PDU. Generally, each type of request PDU has a corresponding type of response PDU. However, if the server determines that a request is improperly formatted or for any reason the server cannot respond with the appropriate PDU type, it will respond with an SDP_ErrorResponse PDU.

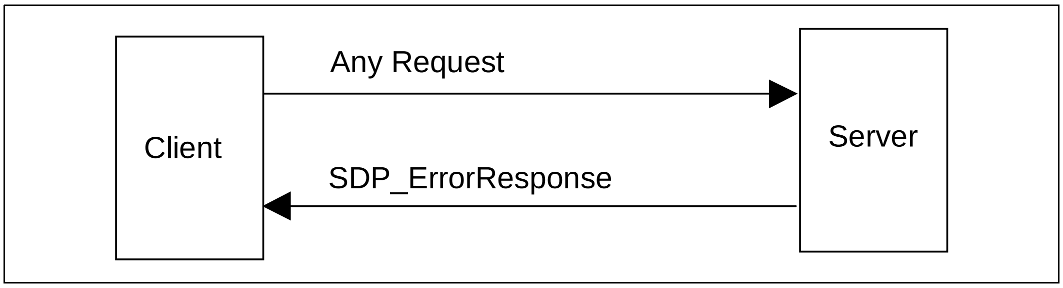


Figure 4.3:

ROKU EXH. 1002

4.4.1 SDP_ErrorResponse PDU

PDU Type	PDU ID	Parameters
SDP_ErrorResponse	0x01	ErrorCode, ErrorInfo

Description:

The SDP server generates this PDU type in response to an improperly formatted request PDU or when the SDP server, for whatever reason, cannot generate an appropriate response PDU.

PDU Parameters:

ErrorCode: *Size: 2 Bytes*

Value	Parameter Description
N	The ErrorCode identifies the reason that an SDP_ErrorResponse PDU was generated.
0x0000	Reserved
0x0001	Invalid/unsupported SDP version
0x0002	Invalid Service Record Handle
0x0003	Invalid request syntax
0x0004	Invalid PDU Size
0x0005	Invalid Continuation State
0x0006	Insufficient Resources to satisfy Request
0x0007-0xFFFF	Reserved

ErrorInfo: *Size: N Bytes*

Value	Parameter Description
Error-specific	ErrorInfo is an ErrorCode-specific parameter. Its interpretation depends on the ErrorCode parameter. The currently defined ErrorCode values do not specify the format of an ErrorInfo field.

4.5 SERVICESEARCH TRANSACTION

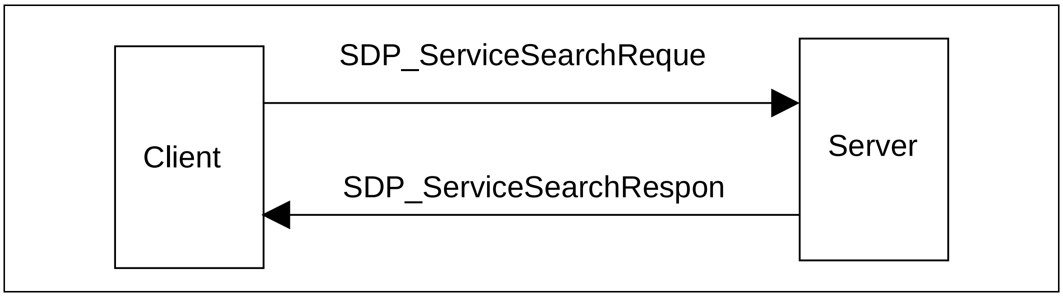


Figure 4.4:

4.5.1 SDP_ServiceSearchRequest PDU

PDU Type	PDU ID	Parameters
SDP_ServiceSearchRequest	0x02	ServiceSearchPattern, MaximumServiceRecordCount, ContinuationState

Description:

The SDP client generates an SDP_ServiceSearchRequest to locate service records that match the service search pattern given as the first parameter of the PDU. Upon receipt of this request, the SDP server will examine its service record data base and return an SDP_ServiceSearchResponse containing the service record handles of service records that match the given service search pattern.

Note that no mechanism is provided to request information for all service records. However, see Section 2.8 Browsing for Services on page 338 for a description of a mechanism that permits browsing for non-specific services without a priori knowledge of the services.

PDU Parameters:

ServiceSearchPattern: *Size: Varies*

Value	Parameter Description
Data Element Sequence	The ServiceSearchPattern is a data element sequence where each element in the sequence is a UUID. The sequence must contain at least one UUID. The maximum number of UUIDs in the sequence is 12[*]. The list of UUIDs constitutes a service search pattern.

*. The value of 12 has been selected as a compromise between the scope of a service search and the size of a search request PDU. It is not expected that more than 12 UUIDs will be useful in a service search pattern.

MaximumServiceRecordCount: *Size: 2 Bytes*

Value	Parameter Description
N	MaximumServiceRecordCount is a 16-bit count specifying the maximum number of service record handles to be returned in the response(s) to this request. The SDP server should not return more handles than this value specifies. If more than N service records match the request, the SDP server determines which matching service record handles to return in the response(s). Range: 0x0001-0xFFFF

ContinuationState: *Size: 1 to 17 Bytes*

Value	**Parameter Description**
Continuation State	ContinuationState consists of an 8-bit count, N, of the number of bytes of continuation state information, followed by the N bytes of continuation state information that were returned in a previous response from the server. N is required to be less than or equal to 16. If no continuation state is to be provided in the request, N is set to 0.

4.5.2 SDP_ServiceSearchResponse PDU

PDU Type	**PDU ID**	**Parameters**
SDP_ServiceSearchResponse	0x03	TotalServiceRecordCount, CurrentServiceRecordCount, ServiceRecordHandleList, ContinuationState

Description:

The SDP server generates an SDP_ServiceSearchResponse upon receipt of a valid SDP_ServiceSearchRequest. The response contains a list of service record handles for service records that match the service search pattern given in the request. Note that if a partial response is generated, it must contain an integral number of complete service record handles; a service record handle value may not be split across multiple PDUs.

PDU Parameters:

TotalServiceRecordCount: *Size: 2 Bytes*

Value	Parameter Description
N	The TotalServiceRecordCount is an integer containing the number of service records that match the requested service search pattern. If no service records match the requested service search pattern, this parameter is set to 0. N should never be larger than the MaximumServiceRecordCount value specified in the SDP_ServiceSearchRequest. When multiple partial responses are used, each partial response contains the same value for TotalServiceRecordCount. Range: 0x0000-0xFFFF

CurrentServiceRecordCount: *Size: 2 Bytes*

Value	Parameter Description
N	The CurrentServiceRecordCount is an integer indicating the number of service record handles that are contained in the next parameter. If no service records match the requested service search pattern, this parameter is set to 0. N should never be larger than the TotalServiceRecordCount value specified in the current response. Range: 0x0000-0xFFFF

ServiceRecordHandleList: *Size: (CurrentServiceRecordCount*4) Bytes*

Value	Parameter Description
List of 32-bit handles	The ServiceRecordHandleList contains a list of service record handles. The number of handles in the list is given in the CurrentServiceRecordCount parameter. Each of the handles in the list refers to a service record that matches the requested service search pattern. Note that this list of service record handles does not have the format of a data element. It contains no header fields, only the 32-bit service record handles.

ContinuationState: *Size: 1 to 17 Bytes*

Value	Parameter Description
Continuation State	ContinuationState consists of an 8-bit count, N, of the number of bytes of continuation state information, followed by the N bytes of continuation information. If the current response is complete, this parameter consists of a single byte with the value 0. If a partial response is contained in the PDU, the ContinuationState parameter may be supplied in a subsequent request to retrieve the remainder of the response.

4.6 SERVICEATTRIBUTE TRANSACTION

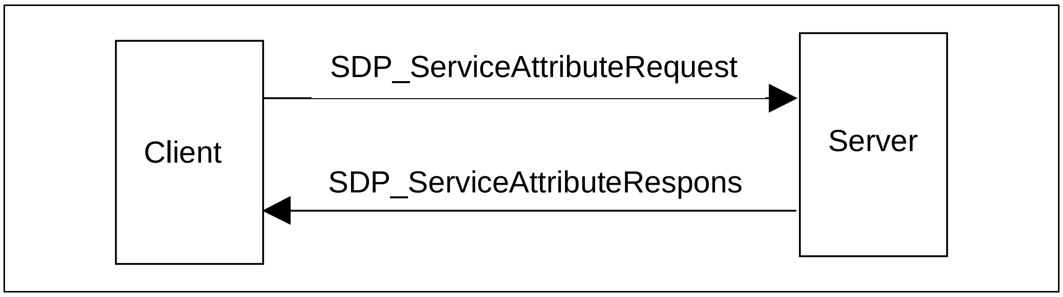


Figure 4.5:

4.6.1 SDP_ServiceAttributeRequest PDU

PDU Type	PDU ID	Parameters
SDP_ServiceAttributeRequest	0x04	ServiceRecordHandle, MaximumAttributeByteCount, AttributeIDList, ContinuationState

Description:

The SDP client generates an SDP_ServiceAttributeRequest to retrieve specified attribute values from a specific service record. The service record handle of the desired service record and a list of desired attribute IDs to be retrieved from that service record are supplied as parameters.

Command Parameters:

ServiceRecordHandle: *Size: 4 Bytes*

Value	Parameter Description
32-bit handle	The ServiceRecordHandle parameter specifies the service record from which attribute values are to be retrieved. The handle is obtained via a previous SDP_ServiceSearch transaction.

MaximumAttributeByteCount: *Size: 2 Bytes*

Value	Parameter Description
N	MaximumAttributeByteCount specifies the maximum number of bytes of attribute data to be returned in the response(s) to this request. The SDP server should not return more than N bytes of attribute data in the response(s). If the requested attributes require more than N bytes, the SDP server determines how to truncate the list. Range: 0x0007-0xFFFF

ROKU EXH. 1002

AttributeIDList: *Size: Varies*

Value	Parameter Description
Data Element Sequence	The AttributeIDList is a data element sequence where each element in the list is either an attribute ID or a range of attribute IDs. Each attribute ID is encoded as a 16-bit unsigned integer data element. Each attribute ID range is encoded as a 32-bit unsigned integer data element, where the high order 16 bits are interpreted as the beginning attribute ID of the range and the low order 16 bits are interpreted as the ending attribute ID of the range. The attribute IDs contained in the AttributeIDList must be listed in ascending order without duplication of any attribute ID values. Note that all attributes may be requested by specifying a range of 0x0000-0xFFFF.

ContinuationState: *Size: 1 to 17 Bytes*

Value	Parameter Description
Continuation State	ContinuationState consists of an 8-bit count, N, of the number of bytes of continuation state information, followed by the N bytes of continuation state information that were returned in a previous response from the server. N is required to be less than or equal to 16. If no continuation state is to be provided in the request, N is set to 0.

4.6.2 SDP_ServiceAttributeResponse PDU

PDU Type	PDU ID	Parameters
SDP_ServiceAttributeResponse	0x05	AttributeListByteCount, AttributeList, ContinuationState

Description:

The SDP server generates an SDP_ServiceAttributeResponse upon receipt of a valid SDP_ServiceAttributeRequest. The response contains a list of attributes (both attribute ID and attribute value) from the requested service record.

PDU Parameters:

AttributeListByteCount: *Size: 2 Bytes*

Value	Parameter Description
N	The AttributeListByteCount contains a count of the number of bytes in the AttributeList parameter. N must never be larger than the MaximumAttributeByteCount value specified in the SDP_ServiceAttributeRequest. Range: 0x0002-0xFFFF

AttributeList: *Size: AttributeListByteCount*

Value	Parameter Description
Data Element Sequence	The AttributeList is a data element sequence containing attribute IDs and attribute values. The first element in the sequence contains the attribute ID of the first attribute to be returned. The second element in the sequence contains the corresponding attribute value. Successive pairs of elements in the list contain additional attribute ID and value pairs. Only attributes that have non-null values within the service record and whose attribute IDs were specified in the SDP_ServiceAttributeRequest are contained in the AttributeList. Neither an attribute ID nor an attribute value is placed in the AttributeList for attributes in the service record that have no value. The attributes are listed in ascending order of attribute ID value.

ContinuationState: *Size: 1 to 17 Bytes*

Value	Parameter Description
Continuation State	ContinuationState consists of an 8-bit count, N, of the number of bytes of continuation state information, followed by the N bytes of continuation information. If the current response is complete, this parameter consists of a single byte with the value 0. If a partial response is given, the ContinuationState parameter may be supplied in a subsequent request to retrieve the remainder of the response.

4.7 SERVICESEARCHATTRIBUTE TRANSACTION

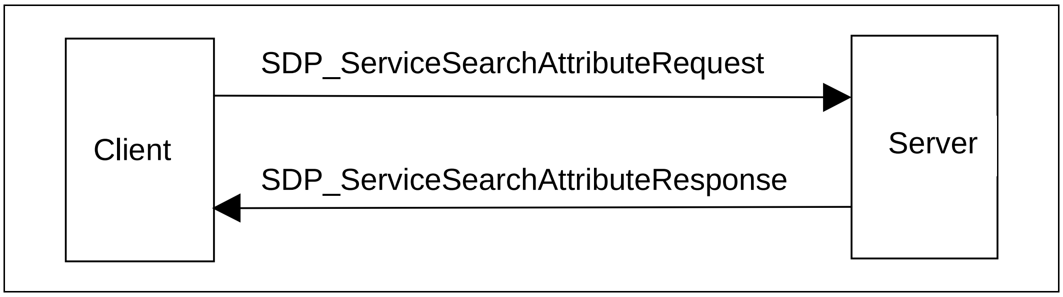

Client
SDP_ServiceSearchAttributeRequest
SDP_ServiceSearchAttributeResponse
Server

Figure 4.6:

4.7.1 SDP_ServiceSearchAttributeRequest PDU

PDU Type	PDU ID	Parameters
SDP_ServiceSearchAttributeRequest	0x06	ServiceSearchPattern, MaximumAttributeByteCount, AttributeIDList, ContinuationState

Description:

The SDP_ServiceSearchAttributeRequest transaction combines the capabilities of the SDP_ServiceSearchRequest and the SDP_ServiceAttributeRequest into a single request. As parameters, it contains both a service search pattern and a list of attributes to be retrieved from service records that match the service search pattern. The SDP_ServiceSearchAttributeRequest and its response are more complex and may require more bytes than separate SDP_ServiceSearch and SDP_ServiceAttribute transactions. However, using SDP_ServiceSearchAttributeRequest may reduce the total number of SDP transactions, particularly when retrieving multiple service records.

Note that the service record handle for each service record is contained in the ServiceRecordHandle attribute of that service and may be requested along with other attributes.

PDU Parameters:

ServiceSearchPattern: *Size: Varies*

Value	Parameter Description
Data Element Sequence	The ServiceSearchPattern is a data element sequence where each element in the sequence is a UUID. The sequence must contain at least one UUID. The maximum number of UUIDs in the sequence is 12[*]. The list of UUIDs constitutes a service search pattern.

*. The value of 12 has been selected as a compromise between the scope of a service search and the size of a search request PDU. It is not expected that more than 12 UUIDs will be useful in a service search pattern.

MaximumAttributeByteCount: *Size: 2 Bytes*

Value	Parameter Description
N	MaximumAttributeByteCount specifies the maximum number of bytes of attribute data to be returned in the response(s) to this request. The SDP server should not return more than N bytes of attribute data in the response(s). If the requested attributes require more than N bytes, the SDP server determines how to truncate the list. Range: 0x0009-0xFFFF

AttributeIDList: *Size: Varies*

Value	Parameter Description
Data Element Sequence	The AttributeIDList is a data element sequence where each element in the list is either an attribute ID or a range of attribute IDs. Each attribute ID is encoded as a 16-bit unsigned integer data element. Each attribute ID range is encoded as a 32-bit unsigned integer data element, where the high order 16 bits are interpreted as the beginning attribute ID of the range and the low order 16 bits are interpreted as the ending attribute ID of the range. The attribute IDs contained in the AttributeIDList must be listed in ascending order without duplication of any attribute ID values. Note that all attributes may be requested by specifying a range of 0x0000-0xFFFF.

ContinuationState: *Size: 1 to 17 Bytes*

Value	Parameter Description
Continuation State	ContinuationState consists of an 8-bit count, N, of the number of bytes of continuation state information, followed by the N bytes of continuation state information that were returned in a previous response from the server. N is required to be less than or equal to 16. If no continuation state is to be provided in the request, N is set to 0.

4.7.2 SDP_ServiceSearchAttributeResponse PDU

PDU Type	PDU ID	Parameters
SDP_ServiceSearchAttributeResponse	0x07	AttributeListsByteCount, AttributeLists, ContinuationState

Description:

The SDP server generates an SDP_ServiceSearchAttributeResponse upon receipt of a valid SDP_ServiceSearchAttributeRequest. The response contains a list of attributes (both attribute ID and attribute value) from the service records that match the requested service search pattern.

PDU Parameters:

AttributeListsByteCount: *Size: 2 Bytes*

Value	Parameter Description
N	The AttributeListsByteCount contains a count of the number of bytes in the AttributeLists parameter. N must never be larger than the MaximumAttributeByteCount value specified in the SDP_ServiceSearchAttributeRequest. Range: 0x0002-0xFFFF

AttributeLists: *Size: Varies*

Value	Parameter Description
Data Element Sequence	The AttributeLists is a data element sequence where each element in turn is a data element sequence representing an attribute list. Each attribute list contains attribute IDs and attribute values from one service record. The first element in each attribute list contains the attribute ID of the first attribute to be returned for that service record. The second element in each attribute list contains the corresponding attribute value. Successive pairs of elements in each attribute list contain additional attribute ID and value pairs. Only attributes that have non-null values within the service record and whose attribute IDs were specified in the SDP_ServiceSearchAttributeRequest are contained in the AttributeLists. Neither an attribute ID nor attribute value is placed in AttributeLists for attributes in the service record that have no value. Within each attribute list, the attributes are listed in ascending order of attribute ID value.

ContinuationState: *Size: 1 to 17 Bytes*

Value	Parameter Description
Continuation State	ContinuationState consists of an 8-bit count, N, of the number of bytes of continuation state information, followed by the N bytes of continuation information. If the current response is complete, this parameter consists of a single byte with the value 0. If a partial response is given, the ContinuationState parameter may be supplied in a subsequent request to retrieve the remainder of the response.

5 SERVICE ATTRIBUTE DEFINITIONS

The service classes and attributes contained in this document are necessarily a partial list of the service classes and attributes supported by SDP. Only service classes that directly support the SDP server are included in this document. Additional service classes will be defined in other documents and possibly in future revisions of this document. Also, it is expected that additional attributes will be discovered that are applicable to a broad set of services; these may be added to the list of Universal attributes in future revisions of this document.

5.1 UNIVERSAL ATTRIBUTE DEFINITIONS

Universal attributes are those service attributes whose definitions are common to all service records. Note that this does not mean that every service record must contain values for all of these service attributes. However, if a service record has a service attribute with an attribute ID allocated to a universal attribute, the attribute value must conform to the universal attribute's definition.

Only two attributes are required to exist in every service record instance. They are the ServiceRecordHandle (attribute ID 0x0000) and the ServiceClassIDList (attribute ID 0x0001). All other service attributes are optional within a service record.

5.1.1 ServiceRecordHandle Attribute

Attribute Name	Attribute ID	Attribute Value Type
ServiceRecordHandle	0x0000	32-bit unsigned integer

Description:

A service record handle is a 32-bit number that uniquely identifies each service record within an SDP server. It is important to note that, in general, each handle is unique only within each SDP server. If SDP server S1 and SDP server S2 both contain identical service records (representing the same service), the service record handles used to reference these identical service records are completely independent. The handle used to reference the service on S1 will, in general, be meaningless if presented to S2.

5.1.2 ServiceClassIDList Attribute

Attribute Name	Attribute ID	Attribute Value Type
ServiceClassIDList	0x0001	Data Element Sequence

Description:

The ServiceClassIDList attribute consists of a data element sequence in which each data element is a UUID representing the service classes that a given service record conforms to. The UUIDs are listed in order from the most specific class to the most general class. The ServiceClassIDList must contain at least one service class UUID.

5.1.3 ServiceRecordState Attribute

Attribute Name	Attribute ID	Attribute Value Type
ServiceRecordState	0x0002	32-bit unsigned integer

Description:

The ServiceRecordState is a 32-bit integer that is used to facilitate caching of ServiceAttributes. If this attribute is contained in a service record, its value is guaranteed to change when any other attribute value is added to, deleted from or changed within the service record. This permits a client to check the value of this single attribute. If its value has not changed since it was last checked, the client knows that no other attribute values within the service record have changed.

5.1.4 ServiceID Attribute

Attribute Name	Attribute ID	Attribute Value Type
ServiceID	0x0003	UUID

Description:

The ServiceID is a UUID that universally and uniquely identifies the service instance described by the service record. This service attribute is particularly useful if the same service is described by service records in more than one SDP server.

5.1.5 ProtocolDescriptorList Attribute

Attribute Name	Attribute ID	Attribute Value Type
ProtocolDescriptorList	0x0004	Data Element Sequence or Data Element Alternative

Description:

The ProtocolDescriptorList attribute describes one or more protocol stacks that may be used to gain access to the service described by the service record.

If the ProtocolDescriptorList describes a single stack, it takes the form of a data element sequence in which each element of the sequence is a protocol descriptor. Each protocol descriptor is, in turn, a data element sequence whose first element is a UUID identifying the protocol and whose successive elements are protocol-specific parameters. Potential protocol-specific parameters are a protocol version number and a connection-port number. The protocol descriptors are listed in order from the lowest layer protocol to the highest layer protocol used to gain access to the service.

If it is possible for more than one kind of protocol stack to be used to gain access to the service, the ProtocolDescriptorList takes the form of a data element alternative where each member is a data element sequence as described in the previous paragraph.

Protocol Descriptors

A protocol descriptor identifies a communications protocol and provides protocol-specific parameters. A protocol descriptor is represented as a data element sequence. The first data element in the sequence must be the UUID that identifies the protocol. Additional data elements optionally provide protocol-specific information, such as the L2CAP protocol/service multiplexer (PSM) and the RFCOMM server channel number (CN) shown below.

ProtocolDescriptorList Examples

These examples are intended to be illustrative. The parameter formats for each protocol are not defined within this specification.

In the first two examples, it is assumed that a single RFCOMM instance exists on top of the L2CAP layer. In this case, the L2CAP protocol specific information (PSM) points to the single instance of RFCOMM. In the last example, two different and independent RFCOMM instances are available on top of the L2CAP layer. In this case, the L2CAP protocol specific information (PSM) points to a distinct identifier that distinguishes each of the RFCOMM instances. According to the L2CAP specification, this identifier takes values in the range 0x1000-0xFFFF.

IrDA-like printer

((L2CAP, PSM=RFCOMM), (RFCOMM, CN=1), (PostscriptStream))

IP Network Printing

((L2CAP, PSM=RFCOMM), (RFCOMM, CN=2), (PPP), (IP), (TCP), (IPP))

Synchronization Protocol Descriptor Example

((L2CAP, PSM=0x1001), (RFCOMM, CN=1), (Obex), (vCal))

((L2CAP, PSM=0x1002), (RFCOMM, CN=1), (Obex), (otherSynchronisationApplication))

5.1.6 BrowseGroupList Attribute

Attribute Name	Attribute ID	Attribute Value Type
BrowseGroupList	0x0005	Data Element Sequence

Description:

The BrowseGroupList attribute consists of a data element sequence in which each element is a UUID that represents a browse group to which the service record belongs. The top-level browse group ID, called PublicBrowseRoot and representing the root of the browsing hierarchy, has the value 00001002-0000-1000-7007-00805F9B34FB (UUID16: 0x1002) from the Bluetooth Assigned Numbers document.

5.1.7 LanguageBaseAttributeIDList Attribute

Attribute Name	Attribute ID	Attribute Value Type
LanguageBaseAttributeIDList	0x0006	Data Element Sequence

Description:

In order to support human-readable attributes for multiple natural languages in a single service record, a base attribute ID is assigned for each of the natural languages used in a service record. The human-readable universal attributes are then defined with an attribute ID offset from each of these base values, rather than with an absolute attribute ID.

The LanguageBaseAttributeIDList attribute is a list in which each member contains a language identifier, a character encoding identifier, and a base attribute

ID for each of the natural languages used in the service record. The Language-BaseAttributeIDList attribute consists of a data element sequence in which each element is a 16-bit unsigned integer. The elements are grouped as triplets (threes).

The first element of each triplet contains an identifier representing the natural language. The language is encoded according to ISO 639:1988 (E/F): “Code for the representation of names of languages”.

The second element of each triplet contains an identifier that specifies a character encoding used for the language. Values for character encoding can be found in IANA's database[2], and have the values that are referred to as MIBEnum values. The recommended character encoding is UTF-8.

The third element of each triplet contains an attribute ID that serves as the base attribute ID for the natural language in the service record. Different service records within a server may use different base attribute ID values for the same language.

To facilitate the retrieval of human-readable universal attributes in a principal language, the base attribute ID value for the primary language supported by a service record must be 0x0100. Also, if a LanguageBaseAttributeIDList attribute is contained in a service record, the base attribute ID value contained in its first element must be 0x0100.

5.1.8 ServiceInfoTimeToLive Attribute

Attribute Name	Attribute ID	Attribute Value Type
ServiceInfoTimeToLive	0x0007	32-bit unsigned integer

Description:

The ServiceTimeToLive attribute is a 32-bit integer that contains the number of seconds for which the information in a service record is expected to remain valid and unchanged. This time interval is measured from the time that the attribute value is retrieved from the SDP server. This value does not imply a guarantee that the service record will remain available or unchanged. It is simply a hint that a client may use to determine a suitable polling interval to re-validate the service record contents.

2. See http://www.isi.edu/in-notes/iana/assignments/character-sets

5.1.9 ServiceAvailability Attribute

Attribute Name	**Attribute ID**	**Attribute Value Type**
ServiceAvailability	0x0008	8-bit unsigned integer

Description:

The ServiceAvailability attribute is an 8-bit unsigned integer that represents the relative ability of the service to accept additional clients. A value of 0xFF indicates that the service is not currently in use and is thus fully available, while a value of 0x00 means that the service is not accepting new clients. For services that support multiple simultaneous clients, intermediate values indicate the relative availability of the service on a linear scale.

For example, a service that can accept up to 3 clients should provide ServiceAvailability values of 0xFF, 0xAA, 0x55, and 0x00 when 0, 1, 2, and 3 clients, respectively, are utilising the service. The value 0xAA is approximately (2/3) * 0xFF and represents 2/3 availability, while the value 0x55 is approximately (1/3)*0xFF and represents 1/3 availability. Note that the availability value may be approximated as

(1 - (current_number_of_clients / maximum_number_of_clients)) * 0xFF

When the maximum number of clients is large, this formula must be modified to ensure that ServiceAvailability values of 0x00 and 0xFF are reserved for their defined meanings of unavailability and full availability, respectively.

Note that the maximum number of clients a service can support may vary according to the resources utilised by the service's current clients.

A non-zero value for ServiceAvailability does not guarantee that the service will be available for use. It should be treated as a hint or an approximation of availability status.

5.1.10 BluetoothProfileDescriptorList Attribute

Attribute Name	**Attribute ID**	**Attribute Value Type**
BluetoothProfileDescriptorList	0x0009	Data Element Sequence

Description:

The BluetoothProfileDescriptorList attribute consists of a data element sequence in which each element is a profile descriptor that contains information about a Bluetooth profile to which the service represented by this service record conforms. Each profile descriptor is a data element sequence whose first element is the UUID assigned to the profile and whose second element is a 16-bit profile version number.

Each version of a profile is assigned a 16-bit unsigned integer profile version number, which consists of two 8-bit fields. The higher-order 8 bits contain the major version number field and the lower-order 8 bits contain the minor version number field. The initial version of each profile has a major version of 1 and a minor version of 0. When upward compatible changes are made to the profile, the minor version number will be incremented. If incompatible changes are made to the profile, the major version number will be incremented.

5.1.11 DocumentationURL Attribute

Attribute Name	Attribute ID	Attribute Value Type
DocumentationURL	0x000A	URL

Description:

This attribute is a URL which points to documentation on the service described by a service record.

5.1.12 ClientExecutableURL Attribute

Attribute Name	Attribute ID	Attribute Value Type
ClientExecutableURL	0x000B	URL

Description:

This attribute contains a URL that refers to the location of an application that may be used to utilize the service described by the service record. Since different operating environments require different executable formats, a mechanism has been defined to allow this single attribute to be used to locate an executable that is appropriate for the client device’s operating environment. In the attribute value URL, the first byte with the value 0x2A (ASCII character ‘*’) is to be replaced by the client application with a string representing the desired operating environment before the URL is to be used.

The list of standardized strings representing operating environments is contained in the Bluetooth Assigned Numbers document.

For example, assume that the value of the ClientExecutableURL attribute is http://my.fake/public/*/client.exe. On a device capable of executing SH3 WindowsCE files, this URL would be changed to http://my.fake/public/sh3-microsoft-wince/client.exe. On a device capable of executing Windows 98 binaries, this URL would be changed to http://my.fake/public/i86-microsoft-win98/client.exe.

5.1.13 IconURL Attribute

Attribute Name	Attribute ID	Attribute Value Type
IconURL	0x000C	URL

Description:

This attribute contains a URL that refers to the location of an icon that may be used to represent the service described by the service record. Since different hardware devices require different icon formats, a mechanism has been defined to allow this single attribute to be used to locate an icon that is appropriate for the client device. In the attribute value URL, the first byte with the value 0x2A (ASCII character ‘*’) is to be replaced by the client application with a string representing the desired icon format before the URL is to be used.

The list of standardized strings representing icon formats is contained in the Bluetooth Assigned Numbers document.

For example, assume that the value of the IconURL attribute is http://my.fake/public/icons/*. On a device that prefers 24 x 24 icons with 256 colors, this URL would be changed to http://my.fake/public/icons/24x24x8.png. On a device that prefers 10 x 10 monochrome icons, this URL would be changed to http://my.fake/public/icons/10x10x1.png.

5.1.14 ServiceName Attribute

Attribute Name	Attribute ID Offset	Attribute Value Type
ServiceName	0x0000	String

Description:

The ServiceName attribute is a string containing the name of the service represented by a service record. It should be brief and suitable for display with an Icon representing the service. The offset 0x0000 must be added to the attribute ID base (contained in the LanguageBaseAttributeIDList attribute) in order to compute the attribute ID for this attribute.

5.1.15 ServiceDescription Attribute

Attribute Name	Attribute ID Offset	Attribute Value Type
ServiceDescription	0x0001	String

Description:

This attribute is a string containing a brief description of the service. It should be less than 200 characters in length. The offset 0x0001 must be added to the attribute ID base (contained in the LanguageBaseAttributeIDList attribute) in order to compute the attribute ID for this attribute.

5.1.16 ProviderName Attribute

Attribute Name	Attribute ID Offset	Attribute Value Type
ProviderName	0x0002	String

Description:

This attribute is a string containing the name of the person or organization providing the service. The offset 0x0002 must be added to the attribute ID base (contained in the LanguageBaseAttributeIDList attribute) in order to compute the attribute ID for this attribute.

5.1.17 Reserved Universal Attribute IDs

Attribute IDs in the range of 0x000D-0x01FF are reserved.

5.2 SERVICEDISCOVERYSERVER SERVICE CLASS ATTRIBUTE DEFINITIONS

This service class describes service records that contain attributes of service discovery server itself. The attributes listed in this section are only valid if the ServiceClassIDList attribute contains the ServiceDiscoveryServerServiceClassID. Note that all of the universal attributes may be included in service records of the ServiceDiscoveryServer class.

5.2.1 ServiceRecordHandle Attribute

Described in the universal attribute definition for ServiceRecordHandle.

Value

A 32-bit integer with the value 0x000000000.

5.2.2 ServiceClassIDList Attribute

Described in the universal attribute definition for ServiceClassIDList.

Value

A UUID representing the ServiceDiscoveryServerServiceClassID.

5.2.3 VersionNumberList Attribute

Attribute Name	Attribute ID	Attribute Value Type
VersionNumberList	0x0200	Data Element Sequence

Description:

The VersionNumberList is a data element sequence in which each element of the sequence is a version number supported by the SDP server.

A version number is a 16-bit unsigned integer consisting of two fields. The higher-order 8 bits contain the major version number field and the low-order 8 bits contain the minor version number field. The initial version of SDP has a major version of 1 and a minor version of 0. When upward compatible changes are made to the protocol, the minor version number will be incremented. If incompatible changes are made to SDP, the major version number will be incremented. This guarantees that if a client and a server support a common major version number, they can communicate if each uses only features of the specification with a minor version number that is supported by both client and server.

5.2.4 ServiceDatabaseState Attribute

Attribute Name	Attribute ID	Attribute Value Type
ServiceDatabaseState	0x0201	32-bit unsigned integer

Description:

The ServiceDatabaseState is a 32-bit integer that is used to facilitate caching of service records. If this attribute exists, its value is guaranteed to change when any of the other service records are added to or deleted from the server's database. If this value has not changed since the last time a client queried its value, the client knows that a) none of the other service records maintained by the SDP server have been added or deleted; and b) any service record handles acquired from the server are still valid. A client should query this attribute's value when a connection to the server is established, prior to using any service record handles acquired during a previous connection.

Note that the ServiceDatabaseState attribute does not change when existing service records are modified, including the addition, removal, or modification of service attributes. A service record's ServiceRecordState attribute indicates when that service record is modified.

5.2.5 Reserved Attribute IDs

Attribute IDs in the range of 0x0202-0x02FF are reserved.

5.3 BROWSEGROUPDESCRIPTOR SERVICE CLASS ATTRIBUTE DEFINITIONS

This service class describes the ServiceRecord provided for each Browse-GroupDescriptor service offered on a Bluetooth device. The attributes listed in this section are only valid if the ServiceClassIDList attribute contains the BrowseGroupDescriptorServiceClassID. Note that all of the universal attributes may be included in service records of the BrowseGroupDescriptor class.

5.3.1 ServiceClassIDList Attribute

Described in the universal attribute definition for ServiceClassIDList.

Value

A UUID representing the BrowseGroupDescriptorServiceClassID.

5.3.2 GroupID Attribute

Attribute Name	Attribute ID	Attribute Value Type
GroupID	0x0200	UUID

Description:

This attribute contains a UUID that can be used to locate services that are members of the browse group that this service record describes.

5.3.3 Reserved Attribute IDs

Attribute IDs in the range of 0x0201-0x02FF are reserved.

APPENDIX A – BACKGROUND INFORMATION

A.1. Service Discovery

As computing continues to move to a network-centric model, finding and making use of services that may be available in the network becomes increasingly important. Services can include common ones such as printing, paging, FAXing, and so on, as well as various kinds of information access such as teleconferencing, network bridges and access points, eCommerce facilities, and so on — most any kind of service that a server or service provider might offer. In addition to the need for a standard way of discovering available services, there are other considerations: getting access to the services (finding and obtaining the protocols, access methods, “drivers” and other code necessary to utilize the service), controlling access to the services, advertising the services, choosing among competing services, billing for services, and so on. This problem is widely recognized; many companies, standards bodies and consortia are addressing it at various levels in various ways. Service Location Protocol (SLP), Jini^{TM}, and Salutation^{TM}, to name just a few, all address some aspect of service discovery.

A.2. Bluetooth Service Discovery

Bluetooth Service Discovery Protocol (SDP) addresses service discovery specifically for the Bluetooth environment. It is optimized for the highly dynamic nature of Bluetooth communications. SDP focuses primarily on discovering services available from or through Bluetooth devices. SDP does not define methods for accessing services; once services are discovered with SDP, they can be accessed in various ways, depending upon the service. This might include the use of other service discovery and access mechanisms such as those mentioned above; SDP provides a means for other protocols to be used along with SDP in those environments where this can be beneficial. While SDP can coexist with other service discovery protocols, it does not require them. In Bluetooth environments, services can be discovered using SDP and can be accessed using other protocols defined by Bluetooth.

APPENDIX B – EXAMPLE SDP TRANSACTIONS

The following are simple examples of typical SDP transactions. These are meant to be illustrative of SDP flows. The examples do not consider:

- Caching (in a caching system, the SDP client would make use of the ServiceRecordState and ServiceDatabaseState attributes);
- Service availability (if this is of interest, the SDP client should use the ServiceAvailability and/or ServiceTimeToLive attributes);
- SDP versions (the VersionNumberList attribute could be used to determine compatible SDP versions);
- SDP Error Responses (an SDP error response is possible for any SDP request that is in error); and
- Communication connection (the examples assume that an L2CAP connection is established).

The examples are meant to be illustrative of the protocol. The format used is `ObjectName[ObjectSizeInBytes] {SubObjectDefinitions}`, but this is not meant to illustrate an interface. The `ObjectSizeInBytes` is the size of the object in decimal. The `SubObjectDefinitions` (inside of `{}` characters) are components of the immediately enclosing object. Hexadecimal values shown as lower-case letters, such as for transaction IDs and service handles, are variables (the particular value is not important for the illustration, but each such symbol always represents the same value). Comments are included in this manner: `/* comment text */`.

B.1. SDP Example 1 – ServiceSearchRequest

The first example is that of an SDP client searching for a generic printing service. The client does not specify a particular type of printing service. In the example, the SDP server has two available printing services. The transaction illustrates:

1. SDP client to SDP server: SDP_ServiceSearchRequest, specifying the PrinterServiceClassID (represented as a DataElement with a 32-bit UUID value of `ppp...ppp`) as the only element of the ServiceSearchPattern. The PrinterServiceClassID is assumed to be a 32-bit UUID and the data element type for it is illustrated. The TransactionID is illustrated as `tttt`.
2. SDP server to SDP client: SDP_ServiceSearchResponse, returning handles to two printing services, represented as qqqqqqqq for the first printing service and `rrrrrrrr` for the second printing service. The Transaction ID is the same value as supplied by the SDP client in the corresponding request (ττττ).

```
/* Sent from SDP Client to SDP server */
SDP_ServiceSearchRequest[15] {
  PDUID[1] {
```

ROKU EXH. 1002

```
    0x02
  }
  TransactionID[2] {
    0xtttt
  }
  ParameterLength[2] {
    0x000A
  }
  ServiceSearchPattern[7] {
    DataElementSequence[7] {
      0b00110 0b101 0x05
      UUID[5] {
        /* PrinterServiceClassID */
        0b00011 0b010 0xpppppppp
      }
    }
  }
  MaximumServiceRecordCount[2] {
    0x0003
  }
  ContinuationState[1] {
    /* no continuation state */
    0x00
  }
}


/* Sent from SDP server to SDP client */
SDP_ServiceSearchResponse[16] {
  PDUID[1] {
    0x03
  }
  TransactionID[2] {
    0xtttt
  }
  ParameterLength[2] {
    0x000D
  }
  TotalServiceRecordCount[2] {
    0x0002
  }
  CurrentServiceRecordCount[2] {
    0x0002
  }
  ServiceRecordHandleList[8] {
    /* print service 1 handle */
    0xqqqqqqqq
    /* print service 2 handle */
    0xrrrrrrrr
  }
  ContinuationState[1] {
    /* no continuation state */
    0x00
  }
}
```

B.2. SDP Example 2 – ServiceAttributeTransaction

The second example continues the first example. In Example 1, the SDP client obtained handles to two printing services. In Example 2, the client uses one of those service handles to obtain the ProtocolDescriptorList attribute for that printing service. The transaction illustrates:

1. SDP client to SDP server: SDP_ServiceAttributeRequest, presenting the previously obtained service handle (the one denoted as qqqqqqqq) and specifying the ProtocolDescriptorList attribute ID (AttributeID 0x0004) as the only attribute requested (other attributes could be retrieved in the same transaction if desired). The TransactionID is illustrated as uuuu to distinguish it from the TransactionID of Example 1.
2. SDP server to SDP client: SDP_ServiceAttributeResponse, returning the ProtocolDescriptorList for the specified printing service. This protocol stack is assumed to be ((L2CAP), (RFCOMM, 2), (PostscriptStream)). The ProtocolDescriptorList is a data element sequence in which each element is, in turn, a data element sequence whose first element is a UUID representing the protocol, and whose subsequent elements are protocol-specific parameters. In this example, one such parameter is included for the RFCOMM protocol, an 8-bit value indicating RFCOMM server channel 2. The Transaction ID is the same value as supplied by the SDP client in the corresponding request (uuuu). The Attributes returned are illustrated as a data element sequence where the protocol descriptors are 32-bit UUIDs and the RFCOMM server channel is a data element with an 8-bit value of 2.

```
 /* Sent from SDP Client to SDP server */
SDP_ServiceAttributeRequest[17] {
  PDUID[1] {
    0x04
  }
  TransactionID[2] {
    0xuuuu
  }
  ParameterLength[2] {
    0x000C
  }
  ServiceRecordHandle[4] {
    0xqqqqqqqq
  }
  MaximumAttributeByteCount[2] {
    0x0080
  }
  AttributeIDList[5] {
    DataElementSequence[5] {
      0b00110 0b101 0x03
      AttributeID[3] {
        0b00001 0b001 0x0004
      }
    }
  }
  ContinuationState[1] {
    /* no continuation state */
    0x00
  }
}

/* Sent from SDP server to SDP client */
SDP_ServiceAttributeResponse[36] {
  PDUID[1] {
    0x05
  }
  TransactionID[2] {
    0xuuuu
  }
  ParameterLength[2] {
    0x0021
  }
  AttributeListByteCount[2] {
    0x001E
  }
  AttributeList[30] {
    DataElementSequence[30] {
      0b00110 0b101 0x1C
      Attribute[28] {
        AttributeID[3] {
          0b00001 0b001 0x0004
        }
        AttributeValue[25] {
          /* ProtocolDescriptorList */
          DataElementSequence[25] {
            0b00110 0b101 0x17
            /* L2CAP protocol descriptor */
            DataElementSequence[7] {
              0b00110 0b101 0x05
              UUID[5] {
                /* L2CAP Protocol UUID */
                0b00011 0b010 <32-bit L2CAP UUID>
              }
            }
            /* RFCOMM protocol descriptor */
            DataElementSequence[9] {
              0b00110 0b101 0x07
              UUID[5] {
                /* RFCOMM Protocol UUID */
                0b00011 0b010 <32-bit RFCOMM UUID>
              }
              /* parameter for server 2 */
              Uint8[2] {
                0b00001 0b000 0x02
              }
            }
            /* PostscriptStream protocol descriptor */
            DataElementSequence[7] {
              0b00110 0b101 0x05
              UUID[5] {
                /* PostscriptStream Protocol UUID */
                0b00011 0b010 <32-bit PostscriptStream UUID>
              }
            }
          }
        }
      }
    }
  }
  ContinuationState[1] {
    /* no continuation state */
    0x00
  }
}
```

B.3. SDP Example 3 – ServiceSearchAttributeTransaction

The third example is a form of service browsing, although it is not generic browsing in that it does not make use of SDP browse groups. Instead, an SDP client is searching for available Synchronization services that can be presented to the user for selection. The SDP client does not specify a particular type of synchronization service. In the example, the SDP server has three available synchronization services: an address book synchronization service and a calendar synchronization service (both from the same provider), and a second calendar synchronization service from a different provider. The SDP client is retrieving the same attributes for each of these services; namely, the data formats supported for the synchronization service (vCard, vCal, ICal, etc.) and those attributes that are relevant for presenting information to the user about the services. Also assume that the maximum size of a response is 400 bytes. Since the result is larger than this, the SDP client will repeat the request supplying a continuation state parameter to retrieve the remainder of the response. The transaction illustrates:

1. SDP client to SDP server: SDP_ServiceSearchAttributeRequest, specifying the generic SynchronisationServiceClassID (represented as a data element whose 32-bit UUID value is `sss...sss`) as the only element of the ServiceSearchPattern. The SynchronisationServiceClassID is assumed to be a 32-bit UUID. The requested attributes are the ServiceRecordHandle (attribute ID 0x0000), ServiceClassIDList (attribute ID 0x0001), IconURL (attribute ID 0x000C), ServiceName (attribute ID 0x0100), ServiceDescription (attribute ID 0x0101), and ProviderName (attributeID 0x0102) attributes; as well as the service-specific SupportedDataStores (AttributeID 0x0301). Since the first two attribute IDs (0x0000 and 0x0001) and three other attribute IDs(0x0100, 0x0101, and 0x0102 are consecutive, they are specified as attribute ranges. The TransactionID is illustrated as vvvv to distinguish it from the TransactionIDs of the other Examples.

   Note that values in the service record's primary language are requested for the text attributes (ServiceName, ServiceDescription and ProviderName) so that absolute attribute IDs may be used, rather than adding offsets to a base obtained from the LanguageBaseAttributeIDList attribute.

2. SDP server to SDP client: SDP_ServiceSearchAttributeResponse, returning the specified attributes for each of the three synchronization services. In the example, each ServiceClassIDList is assumed to contain a single element, the generic SynchronisationServiceClassID (a 32-bit UUID represented as sss...sss). Each of the other attributes contain illustrative data in the example (the strings have illustrative text; the icon URLs are illustrative, for each of the respective three synchronization services; and the SupportedDataStore attribute is represented as an unsigned 8-bit integer where 0x01 = vCard2.1, 0x02 = vCard3.0, 0x03 = vCal1.0 and 0x04 = iCal). Note that one of the service records (the third for which data is returned) has no ServiceDescription attribute. The attributes are returned as a data element sequence, where each element is in turn a data element sequence repre-

senting a list of attributes. Within each attribute list, the ServiceClassIDList is a data element sequence while the remaining attributes are single data elements. The Transaction ID is the same value as supplied by the SDP client in the corresponding request (ϖϖϖϖ). Since the entire result cannot be returned in a single response, a non-null continuation state is returned in this first response.

Note that the total length of the initial data element sequence (487 in the example) is indicated in the first response, even though only a portion of this data element sequence (368 bytes in the example, as indicated in the AttributeLists byte count) is returned in the first response. The remainder of this data element sequence is returned in the second response (without an additional data element header).

3. SDP client to SDP server: SDP_ServiceSearchAttributeRequest, with the same parameters as in step 1, except that the continuation state received from the server in step 2 is included as a request parameter. The TransactionID is changed to ωωωω to distinguish it from previous request.
4. SDP server to SDP client: SDP_ServiceSearchAttributeResponse, with the remainder of the result computed in step 2 above. Since all of the remaining result fits in this second response, a null continuation state is included.

```
/* Part 1 -- Sent from SDP Client to SDP server */
SdpSDP_ServiceSearchAttributeRequest[33] {
  PDUID[1] {
    0x06
  }
  TransactionID[2] {
    0xvvvv
  }
  ParameterLength[2] {
    0x001B
  }
  ServiceSearchPattern[7] {
    DataElementSequence[7] {
      0b00110 0b101 0x05
      UUID[5] {
        /* SynchronisationServiceClassID */
        0b00011 0b010 0xssssssss
      }
    }
  }
  MaximumAttributeByteCount[2] {
    0x0190
  }
AttributeIDList[18] {
    DataElementSequence[18] {
      0b00110 0b101 0x10
      AttributeIDRange[5] {
        0b00001 0b010 0x00000001
      }
      AttributeID[3] {
        0b00001 0b001 0x000C
      }
      AttributeIDRange[5] {
        0b00001 0b010 0x01000102
      }
      AttributeID[3] {
        0b00001 0b001 0x0301
      }    }
  }
  ContinuationState[1] {
    /* no continuation state */
    0x00
  }
}

/* Part 2 -- Sent from SDP server to SDP client */
SdpSDP_ServiceSearchAttributeResponse[384] {
  PDUID[1] {
    0x07
  }
  TransactionID[2] {
    0xvvvv
  }
  ParameterLength[2] {
    0x017B
  }
  AttributeListByteCount[2] {
    0x0170
  }
  AttributeLists[368] {
    DataElementSequence[487] {
      0b00110 0b110 0x01E4
      DataElementSequence[178] {
        0b00110 0b101 0xB0
        Attribute[8] {
          AttributeID[3] {
            0b00001 0b001 0x0000
          }
          AttributeValue[5] {
            /* service record handle */
            0b00001 0b010 0xhhhhhhhh
          }
        }
        Attribute[10] {
          AttributeID[3] {
            0b00001 0b001 0x0001
          }
AttributeValue[7] {
            DataElementSequence[7] {
              0b00110 0b101 0x05
              UUID[5] {
                /* SynchronisationServiceClassID */
                0b00011 0b010 0xssssssss
              }
            }
          }
        }
        Attribute[35] {
          AttributeID[3] {
            0b00001 0b001 0x000C
          }
          AttributeValue[32] {
            /* IconURL; '*' replaced by client application */
            0b01000 0b101 0x1E
            "http://Synchronisation/icons/*"
          }
        }
        Attribute[22] {
          AttributeID[3] {
            0b00001 0b001 0x0100
          }
          AttributeValue[19] {
            /* service name */
            0b00100 0b101 0x11
            "Address Book Sync"
          }
        }
        Attribute[59] {
          AttributeID[3] {
            0b00001 0b001 0x0101
          }
          AttributeValue[56] {
            /* service description */
            0b00100 0b101 0x36
            "Synchronisation Service for"
            " vCard Address Book Entries"
          }
        }
        Attribute[37] {
          AttributeID[3] {
            0b00001 0b001 0x0102
          }
          AttributeValue[34] {
            /* service provider */
            0b00100 0b101 0x20
            "Synchronisation Specialists Inc."
          }
        }
        Attribute[5] {
          AttributeID[3] {
            0b00001 0b001 0x0301
          }
          AttributeValue[2] {
            /* Supported Data Store 'phonebook' */
            0b00001 0b000 0x01
          }
        }
      }
      DataElementSequence[175] {
        0b00110 0b101 0xAD
        Attribute[8] {
          AttributeID[3] {
            0b00001 0b001 0x0000
          }
          AttributeValue[5] {
```

```
      /* service record handle */
      0b00001 0b010 0xmmmmmmmm
    }
  }
  Attribute[10] {
    AttributeID[3] {
      0b00001 0b001 0x0001
    }
              AttributeValue[7] {
      DataElementSequence[7] {
        0b00110 0b101 0x05
        UUID[5] {
          /* SynchronisationServiceClassID */
          0b00011 0b010 0xssssssss
        }
      }
    }
  }
  Attribute[35] {
    AttributeID[3] {
      0b00001 0b001 0x000C
    }
    AttributeValue[32] {
      /* IconURL; '*' replaced by client application */
      0b01000 0b101 0x1E
      "http://Synchronisation/icons/*"
    }
  }
  Attribute[21] {
    AttributeID[3] {
      0b00001 0b001 0x0100
    }
    AttributeValue[18] {
      /* service name */
      0b00100 0b101 0x10
      "Appointment Sync"
    }
  }
  Attribute[57] {
    AttributeID[3] {
      0b00001 0b001 0x0101
    }
    AttributeValue[54] {
      /* service description */
      0b00100 0b101 0x34
      "Synchronisation Service for"
      " vCal Appointment Entries"
    }
  }
  Attribute[37] {
    AttributeID[3] {
      0b00001 0b001 0x0102
    }
    AttributeValue[34] {
      /* service provider */
      0b00100 0b101 0x20
      "Synchronisation Specialists Inc."
```

```
          }
        }
        Attribute[5] {
          AttributeID[3] {
            0b00001 0b001 0x0301
          }
          AttributeValue[2] {
            /* Supported Data Store ’calendar’ */
            0b00001 0b000 0x03
          }
        }
      }
    /* } Data element sequence of attribute lists */
    /* is not completed in this PDU. */
  }
  ContinuationState[9] {
    /* 8 bytes of continuation state */
    0x08 0xzzzzzzzzzzzzzzzz
  }
}

/* Part 3 -- Sent from SDP Client to SDP server */
SdpSDP_ServiceSearchAttributeRequest[41] {
  PDUID[1] {
    0x06
  }
  TransactionID[2] {
    0xwwww
  }
  ParameterLength[2] {
    0x0024
  }
  ServiceSearchPattern[7] {
    DataElementSequence[7] {
      0b00110 0b101 0x05
      UUID[5] {
        /* SynchronisationServiceClassID */
        0b00011 0b010 0xssssssss
      }
    }
  }
  MaximumAttributeByteCount[2] {
    0x0180
  }
    AttributeIDList[18] {
    DataElementSequence[18] {
      0b00110 0b101 0x10
      AttributeIDRange[5] {
        0b00001 0b010 0x00000001
      }
      AttributeID[3] {
        0b00001 0b001 0x000C
      }
      AttributeIDRange[5] {
        0b00001 0b010 0x01000102
      }
      AttributeID[3] {
```

ROKU EXH. 1002

```
        0b00001 0b001 0x0301
      }
    }
  }
  ContinuationState[9] {
    /* same 8 bytes of continuation state */
    /* received in part 2 */
    0x08 0xzzzzzzzzzzzzzzzz
  }
}

Part 4 -- Sent from SDP server to SDP client

SdpSDP_ServiceSearchAttributeResponse[115] {
  PDUID[1] {
    0x07
  }
  TransactionID[2] {
    0xwwww
  }
  ParameterLength[2] {
    0x006E
  }
  AttributeListByteCount[2] {
    0x006B
  }
  AttributeLists[107] {
    /* Continuing the data element sequence of */
    /* attribute lists begun in Part 2. */
      DataElementSequence[107] {
        0b00110 0b101 0x69
        Attribute[8] {
          AttributeID[3] {
            0b00001 0b001 0x0000
          }
          AttributeValue[5] {
            /* service record handle */
            0b00001 0b010 0xffffffff
          }
        }
        Attribute[10] {
          AttributeID[3] {
            0b00001 0b001 0x0001
          }
          AttributeValue[7] {
            DataElementSequence[7] {
              0b00110 0b101 0x05
              UUID[5] {
                /* SynchronisationServiceClassID */
                0b00011 0b010 0xssssssss
              }
            }
          }
        }
        Attribute[35] {
          AttributeID[3] {
            0b00001 0b001 0x000C
          }
          AttributeValue[32] {
            /* IconURL; '*' replaced by client application */
            0b01000 0b101 0x1E
            "http://DevManufacturer/icons/*"
          }
        }
        Attribute[18] {
          AttributeID[3] {
            0b00001 0b001 0x0100
          }
          AttributeValue[15] {
            /* service name */
            0b00100 0b101 0x0D
            "Calendar Sync"
          }
        }
        Attribute[29] {
          AttributeID[3] {
            0b00001 0b001 0x0102
          }
          AttributeValue[26] {
            /* service provider */
            0b00100 0b101 0x18
            "Device Manufacturer Inc."
          }
        }
        Attribute[5] {
          AttributeID[3] {
            0b00001 0b001 0x0301
          }
          AttributeValue[2] {
            /* Supported Data Store 'calendar' */
            0b00001 0b000 0x03
          }
        }
      }
    /* This completes the data element sequence */
    /* of attribute lists begun in Part 2.
  }
  ContinuationState[1] {
    /* no continuation state */
    0x00
  }
}
```

APPENDIX Q

Internet Engineering Task Force
INTERNET DRAFT

Yaron Y. Goland
Ting Cai
Paul Leach
Ye Gu
Microsoft Corporation
Shivaun Albright
Hewlett-Packard Company

October 28, 1999
Expires April 2000

Simple Service Discovery Protocol/1.0
Operating without an Arbiter
<draft-cai-ssdp-v1-03.txt>

Status of this Memo

Please send comments to the SSDP mailing list. Subscription information for the SSDP mailing list is available at http://www.upnp.org/resources/ssdpmail.htm.

Abstract

The Simple Service Discovery Protocol (SSDP) provides a mechanism where by network clients, with little or no static configuration, can discover network services. SSDP accomplishes this by providing for multicast discovery support as well as server based notification and discovery routing.

Table of Contents

Status of this Memo...1
Abstract...1

Table of Contents...1
1. Changes Since 02...3
2. Introduction...3
  2.1. Problem Statement..3
  2.2. Proposed Solution..4
    2.2.1. Message Flow on the SSDP Multicast Channel...........4
    2.2.2. SSDP Discovery Information Caching Model.............4
  2.3. Design Rationale...5
    2.3.1. Message Flow on the SSDP Multicast Channel...........5
    2.3.2. SSDP Discovery Information Caching Model.............7
3. Terminology..8
4. SSDP Discovery Requests......................................8
  4.1. Problem Statement..8
  4.2. Proposed Solution..8
  4.3. Design Rationale..10
    4.3.1. Why is the ST header so limited? Why doesn't it support at least and/or/not? Why not name/value pair searching?.....10
    4.3.2. If we are using the SEARCH method why aren't you using the DASL search syntax?.................................10
    4.3.3. Why can we only specify one search type in the ST header of a ssdp:discover request?.........................10
    4.3.4. Why do we only provide support for multicast UDP, not TCP, ssdp:discover requests?.............................10
    4.3.5. Why do we require that responses without caching information not be cached at all?..........................11
5. SSDP Presence Announcements.................................11
  5.1. Problem Statement.......................................11
  5.2. Proposed Solution.......................................11
    5.2.1. ssdp:alive...11
    5.2.2. ssdp:byebye..12
  5.3. Design Rationale..13
    5.3.1. Why are we using GENA NOTIFY requests?.............13
    5.3.2. Why is there no response to the ssdp:alive/ssdp:byebye requests sent to the SSDP multicast channel/port?..........13
    5.3.3. Could NTS values other than ssdp:alive/ssdp:byebye be sent to the SSDP multicast channel/port?...................13
    5.3.4. Why do we include the NT header on ssdp:byebye requests?...13
    5.3.5. Shouldn't the NT and NTS values be switched?........13
6. SSDP Auto-Shut-Off Algorithm................................13
  6.1. Problem Statement.......................................13
  6.2. Proposed Solution.......................................13
  6.3. Design Rationale..14
    6.3.1. Why do we need an auto-shut-off algorithm?..........14
    6.3.2. Why not just require everyone to support directories and thus get around the scaling issue?......................15
7. ssdp:all..15
  7.1. Problem Statement.......................................15
  7.2. Proposed Solution.......................................15
  7.3. Design Rationale..16
    7.3.1. Why would anyone want to enumerate all services?....16
8. SSDP Reserved Multicast Channel.............................16

8.1. Problem Statement...16
8.2. Proposed Solution...16
8.3. Design Rationale..16
8.3.1. Why didn't SSDP just get a static local administrative scope address rather than a relative address?...............16
8.3.2. Why does SSDP need to use a port other than 80?.....16
9. HTTP Headers..17
9.1. USN Header..17
9.2. ST Header...17
10. Security Considerations....................................17
11. IANA Considerations..17
12. Appendix - Constants.......................................17
13. Acknowledgements...17
14. References...17
15. Author's Addresses...18

1. Changes Since 02

The entire specification has been extensively re-written. As such the reader is advised to re-read the entire specification rather than to just look for particular changes.

Removed the arbiter and related functionality.

Spec used to contain both ssdp:discover and ssdp:discovery, settled on ssdp:discover.

Changed SSDP multicast message examples to use the reserved relative multicast address "5" provided by IANA. In the local administrative scope, the only scope currently used by SSDP, this address translates to 239.255.255.250.

An application has been made for a reserved port for SSDP but no response from IANA has been received.

2. Introduction

[Ed. Note: In my experience, one of the best ways to enable a specification to be quickly and successfully developed is to provide a problem statement, a proposed solution and a design rationale. I came across this three-part design structure when Larry Masinter proposed it to the WebDAV WG. To that end, I have divided this spec in a similar manner. Once the specification is sufficiently mature, the problem statement and design rationale sections will be placed in a separate document and the proposed solutions will be presented for standardization.]

This document assumes the reader is very familiar with [RFC2616], [HTTPUDP], [GENA], [MAN] and [RFC2365].

2.1. Problem Statement

ROKU EXH. 1002

A mechanism is needed to allow HTTP clients and HTTP resources to discover each other in local area networks. That is, a HTTP client may need a particular service that may be provided by one or more HTTP resources. The client needs a mechanism to find out which HTTP resources provide the service the client desires.

For the purposes of this specification the previously mentioned HTTP client will be referred to as a SSDP client. The previous mentioned HTTP resource will be referred to as a SSDP service.

In the simplest case this discovery mechanism needs to work without any configuration, management or administration. For example, if a user sets up a home network or a small company sets up a local area network they must not be required to configure SSDP before SSDP can be used to help them discover SSDP services in the form of Printers, Scanners, Fax Machines, etc.

It is a non-goal for SSDP to provide for multicast scope bridging or for advanced query facilities.

2.2. Proposed Solution

2.2.1. Message Flow on the SSDP Multicast Channel

The following is an overview of the messages used to implement SSDP.

SSDP clients discover SSDP services using the reserved local administrative scope multicast address 239.255.255.250 over the SSDP port [NOT YET ALLOCATED BY IANA].

For brevity’s sake the SSDP reserved local administrative scope multicast address and port will be referred to as the SSDP multicast channel/Port.

Discovery occurs when a SSDP client multicasts a HTTP UDP discovery request to the SSDP multicast channel/Port. SSDP services listen to the SSDP multicast channel/Port in order to hear such discovery requests. If a SSDP service hears a HTTP UDP discovery request that matches the service it offers then it will respond using a unicast HTTP UDP response.

SSDP services may send HTTP UDP notification announcements to the SSDP multicast channel/port to announce their presence.

Hence two types of SSDP requests will be sent across the SSDP multicast channel/port. The first are discovery requests, a SSDP client looking for SSDP services. The second are presence announcements, a SSDP service announcing its presence.

2.2.2. SSDP Discovery Information Caching Model

The following provides an overview of the data provided in a SSDP system.

Services are identified by a unique pairing of a service type URI and a Unique Service Name (USN) URI.

Service types identify a type of service, such as a refrigerator, clock/radio, what have you. The exact meaning of a service type is outside the scope of this specification. For the purposes of this specification, a service type is an opaque identifier that identifies a particular type of service.

A USN is a URI that uniquely identifies a particular instance of a service. USNs are used to differentiate between two services with the same service type.

In addition to providing both a service type and a USN, discovery results and presence announcements also provide expiration and location information.

Location information identifies how one should contact a particular service. One or more location URIs may be included in a discovery response or a presence announcement.

Expiration information identifies how long a SSDP client should keep information about the service in its cache. Once the entry has expired it is to be removed from the SSDP client's cache.

Thus a SSDP client service cache might look like:

USN URI	Service Type URI	Expiration	Location
upnp:uuid:k91...	upnp:clockradio	3 days	http://foo.com/cr
uuid:x7z...	ms:wince	1 week	http://msce/win

In the previous example both USN URIs are actually UUIDs such as upnp:uuid:k91d4fae-7dec-11d0-a765-00a0c91c6bf6.

If an announcement or discovery response is received that has a USN that matches an entry already in the cache then the information in the cache is to be completely replaced with the information in the announcement or discovery response.

2.3. Design Rationale

[Ed. Note: In my own experience one of the most powerful ways to explain design rationale is in a question/answer form. Therefore I have used that format here.]

2.3.1. Message Flow on the SSDP Multicast Channel

Please see section 8.3 for more design rationale behind our use of multicasting.

2.3.1.1. Why use multicast for communication?

We needed a solution for communication that would work even if there was no one around to configure things. The easiest solution would have been to build a discovery server, but who would set the server up? Who would maintain it? We needed a solution that could work even if no one had any idea what discovery was. By using multicasting we have the equivalent of a "party channel." Everyone can just grab the channel and scream out what they need and everyone else will hear. This means no configuration worries. Of course it brings up other problems which are addressed throughout this specification.

2.3.1.2. Why use a local administrative scope multicast address?

Multicasting comes in many scopes, from link local all the way to "the entire Internet." Our goal is to provide for discovery for local area networks not for the entire Internet. LANs often are bridged/routed so a link local multicast scope was too restrictive. The next level up was a local administrative scope. The idea being that your administrator decides how many machines should be grouped together and considered a "unit". This seemed the ideal scope to use for a local discovery protocol.

2.3.1.3. Why does SSDP support both service discovery requests as well as service presence announcements?

Some discovery protocols only support discovery requests, that is, the client must send out a request in order to find out who is around. The downside to such solutions is that they tend to be very expensive on the wire. For example, we want to display to our user all the VCRs in her house. So we send out a discovery request. However our user has just purchased a new VCR and, after starting our program, plugged it in. The only way we would find out about the new VCR and be able to display it on our user’s screen is by constantly sending out discovery requests. Now imagine every client in the network having to send out a torrent of discovery requests for service they care about in order to make sure they don’t miss a new service coming on-line.

Other systems use the opposite extreme, they only support announcements. Therefore, when our user opens the VCR display window we would just sit and listen for announcements. In such systems all the services have to send out a constant stream of announcements in order to make sure that no one misses them. Users aren’t the most patient people in the world so each service will probably need to announce itself at least every few seconds. This constant stream of traffic does horrible things to network efficient, especially for shared connections like Ethernets.

SSDP decided to adopt a hybrid approach and do both discovery and announcements. This can be incredibly efficient. When a service first comes on-line it will send out an announcement so that everyone knows it is there. At that point it shouldn’t ever need to send out another announcement unless it is going off-line, has changed state or its cache entry is about to expire. Any clients who come on-line after the service came on-line will discover the desired service by sending out a discovery request. The client should never need to repeat the discovery request because any services that subsequently come on-line will announce themselves. The end result is that no one needs to send out steady streams of messages. The entire system is event driven, only when things change will messages need to be sent out. The cost, however, is that the protocol is more complex. We felt this was a price worth paying as it meant that SSDP could be used successfully in fairly large networks.

2.3.1.4. Doesn’t the caching information turn SSDP back into a "announcement driven" protocol?

Discovery protocols that only support announcements generally have to require services to send announcements every few seconds. Otherwise users screens will take too long to update with information about which services are available.

SSDP, on the other hand, allows the service to inform clients how long they should assume the service is around. Thus a service can set a service interval to seconds, minutes, days, weeks, months or even years.

Clients do not have to wait around for cache update messages because they can perform discovery.

2.3.2. SSDP Discovery Information Caching Model

2.3.2.1. Why do we need USNs, isn’t the location good enough?

When a service announces itself it usually includes a location identifying where it may be found. However that location can and will change over time. For example, a user may decide to change the DNS name assigned to that device. Were we to depend on locations, not USNs, when the service’s location was changed we would think we were seeing a brand new service. This would be very disruptive to the user’s experience. Imagine, for example, that the user has set up a PC program that programs their VCR based on schedules pulled off the Internet. If the user decides to change the VCR’s name from the factory default to something friendly then a location based system would loose track of the VCR it is supposed to be programming because the name has changed. By using unique Ids instead we are able to track the VCR regardless of the name change. So the user can change the VCR's name at will and the VCR programming application will still be able to program the correct VCR.

2.3.2.2. Why are USNs URIs and why are they required to be unique across the entire URI namespace for all time?

In general making a name universally unique turns out to usually be a very good idea. Mechanisms such as UUIDs allow universally unique names to be cheaply created in a decentralized manner. In this case making USNs globally unique is very useful because services may be constantly moved around, if they are to be successfully tracked they need an identifier that isn't going to change and isn't going to get confused with any other service.

URIs were chosen because they have become the de facto managed namespace for use on the Internet. Anytime someone wants to name something it is easy to just use a URI.

3. Terminology

SSDP Client - A HTTP client that makes use of a service.

SSDP Service - A HTTP resource that provides a service used by SSDP clients.

Service Type - A URI that identifies the type or function of a particular service.

Unique Service Name (USN) - A URI that is guaranteed to be unique across the entire URI namespace for all time. It is used to uniquely identify a particular service in order to allow services with identical service type URIs to to be differentiated.

In addition, the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

4. SSDP Discovery Requests

4.1. Problem Statement

A mechanism is needed for SSDP clients to find desired SSDP services.

4.2. Proposed Solution

The SEARCH method, introduced by [DASL], is extended using the [MAN] mechanism to provide for SSDP discovery.

The SSDP SEARCH extension is identified by the URI ssdp:discover.

For brevity's sake a HTTP SEARCH method enhanced with the ssdp:discover functionality will be referred to as a ssdp:discover request.

ssdp:discover requests MUST contain a ST header. ssdp:discover requests MAY contain a body but the body MAY be ignored if not understood by the HTTP service.

The ST header contains a single URI. SSDP clients may use the ST header to specify the service type they want to discover.

This specification only specifies the use of ssdp:discover requests over HTTP Multicast UDP although it is expected that future specifications will expand the definition to handle ssdp:discover requests sent over HTTP TCP.

ssdp:discover requests sent to the SSDP multicast channel/port MUST have a request-URI of "*". Note that future specifications may allow for other request-URIs to be used so implementations based on this specification MUST be ready to ignore ssdp:discover requests on the SSDP multicast channel/port with a request-URI other than "*".

Only SSDP services that have a service type that matches the value in the ST header MAY respond to a ssdp:discover request on the SSDP multicast channel/port.

Responses to ssdp:discover requests sent over the SSDP multicast channel/port are to be sent to the IP address/port the ssdp:discover request came from.

A response to a ssdp:discover request SHOULD include the service's location expressed through the Location and/or AL header. A successful response to a ssdp:discover request MUST also include the ST and USN headers.

Response to ssdp:discover requests SHOULD contain a cache-control: max-age or Expires header. If both are present then they are to be processed in the order specified by HTTP/1.1, that is, the cache-control header takes precedence of the Expires header. If neither the cache-control nor the Expires header is provided on the response to a ssdp:discover request then the information contained in that response MUST NOT be cached by SSDP clients.

4.2.1.1. Example

```
M-SEARCH * HTTP/1.1
S: uuid:ijklmnop-7dec-11d0-a765-00a0c91e6bf6
Host: 239.255.255.250:reservedSSDPport
Man: "ssdp:discover"
ST: ge:fridge
MX: 3
```

```
   HTTP/1.1 200 OK
   S: uuid:ijklmnop-7dec-11d0-a765-00a0c91e6bf6
   Ext:
   Cache-Control: no-cache="Ext", max-age = 5000
   ST: ge:fridge
   USN: uuid:abcdefgh-7dec-11d0-a765-00a0c91e6bf6
   AL: <blender:ixl><http://foo/bar>
```

4.3. Design Rationale

4.3.1. Why is the ST header so limited? Why doesn't it support at least and/or/not? Why not name/value pair searching?

Deciding the "appropriate" level of search capability is a hopeless task. So we decided to pare things back to the absolute minimum, a single opaque token, and see what happens. The result so far has been a very nice, simple, easy to implement, easy to use discovery system. There are lots of great features it doesn't provide but most of them, such as advanced queries and scoping, require a search engine and a directory. This level of capability is beyond many simple devices, exactly the sort of folks we are targeting with SSDP. Besides, search functionality seems to be an all or nothing type of situation. Either you need a brain dead simple search mechanism or you need a full fledged near SQL class search system. Instead of making SSDP the worst of both worlds we decided to just focus on the dirt simple search problem and leave the more advanced stuff to the directory folk.

4.3.2. If we are using the SEARCH method why aren't you using the DASL search syntax?

We couldn't come up with a good reason to force our toaster ovens to learn XML. The features the full-fledged DASL search syntax provides are truly awesome and thus way beyond our simple scenarios. We fully expect that DASL will be the preferred solution for advanced search scenarios, but that isn't what this draft is about.

4.3.3. Why can we only specify one search type in the ST header of a ssdp:discover request?

We wanted to start as simple as possible and be forced, kicking and screaming, into adding additional complexity. The simplest solution was to only allow a single value in the ST header. We were also concerned that if we allowed multiple values into the ST headers somebody would try to throw in and/or/not functionality. Given the minimal byte savings of allowing multiple values into the ST header it seems better to just leave the protocol simpler.

4.3.4. Why do we only provide support for multicast UDP, not TCP, ssdp:discover requests?

We only define what we need to make the discovery protocol work and we don't need TCP to make the discovery protocol work. Besides to make TCP discovery really work you need to be able to handle compound responses which means you need a compound response format which is probably XML and that is more than we wanted to handle. Eventually we expect that you will be able to go up to the SSDP port on a server using a HTTP TCP request and discover what service, if any, lives there. But that will be described in a future specification.

4.3.5. Why do we require that responses without caching information not be cached at all?

Because that was a lot easier thing to do then trying to explain the various heuristics one could use to deal with services who don't provide caching information.

5. SSDP Presence Announcements

5.1. Problem Statement

A mechanism is needed for SSDP services to be able to let interested SSDP clients know of their presence.

A mechanism is needed to allow SSDP services to update expiration information in cache entries regarding them.

A mechanism is needed to allow SSDP services to notify interested SSDP clients when their location changes.

A mechanism is needed to allow SSDP services to inform interested SSDP clients that they are going to de-activate themselves.

5.2. Proposed Solution

5.2.1. ssdp:alive

SSDP services may declare their presence on the network by sending a [GENA] NOTIFY method using the NTS value ssdp:alive to the SSDP multicast channel/port.

For brevity's sake HTTP NOTIFY methods with the NTS value ssdp:alive will be referred to as ssdp:alive requests.

When a ssdp:alive request is received whose USN matches the USN of an entry already in the SSDP client's cache then all information regarding that USN is to be replaced with the information on the ssdp:alive request. Hence ssdp:alive requests can be used to update location information and prevent cache entries from expiring.

The value of NT on a ssdp:alive request MUST be set to the service’s service type. ssdp:alive requests MUST contain a USN header set to the SSDP service’s USN.

ssdp:alive requests SHOULD contain a Location and/or AL header. If there is no DNS support available on the local network then at least one location SHOULD be provided using an IP address of the SSDP service.

ssdp:alive requests SHOULD contain a cache-control: max-age or Expires header. If both are present then they are to be processed in the order specified by HTTP/1.1, that is, the cache-control header takes precedence of the Expires header. If neither the cache-control nor the Expires header is provided the information in the ssdp:alive request MUST NOT be cached by SSDP clients.

There is no response to a ssdp:alive sent to the SSDP multicast channel/port.

5.2.1.1. Example

```
NOTIFY * HTTP/1.1
Host: 239.255.255.250:reservedSSDPport
NT: blenderassociation:blender
NTS: ssdp:alive
USN: someunique:idscheme3
AL: <blender:ixl><http://foo/bar>
Cache-Control: max-age = 7393
```

5.2.2. ssdp:byebye

SSDP services may declare their intention to cease operating by sending a [GENA] NOTIFY method using the NTS value ssdp:byebye to the SSDP multicast channel/port.

For brevity’s sake HTTP NOTIFY methods with the NTS value ssdp:byebye will be referred to as ssdp:byebye requests.

The value of NT on a ssdp:byebye request MUST be set to the service’s service type. ssdp:byebye requests MUST contain a USN header set to the SSDP service’s USN.

There is no response to a ssdp:byebye sent to the SSDP multicast channel/port.

When a ssdp:byebye request is received all cached information regarding that USN SHOULD be removed.

5.2.2.1. Example

```
NOTIFY * HTTP/1.1
Host: 239.255.255.250:reservedSSDPport
```

```
   NT: someunique:idscheme3
   NTS: ssdp:byebye
   USN: someunique:idscheme3
```

5.3. Design Rationale

5.3.1. Why are we using GENA NOTIFY requests?

We needed to use some notification format and GENA seemed as good as any. Moving forward, GENA gives us a framework to do notification subscriptions which will be necessary if SSDP services are to be able to provide status updates across the wilds of the Internet without depending on the largely non-existent Internet multicast infrastructure.

5.3.2. Why is there no response to the ssdp:alive/ssdp:byebye requests sent to the SSDP multicast channel/port?

What response would be sent? There isn't much of a point of having the SSDP clients send response saying "we received your notification" since there may be a lot of them.

5.3.3. Could NTS values other than ssdp:alive/ssdp:byebye be sent to the SSDP multicast channel/port?

Yes.

5.3.4. Why do we include the NT header on ssdp:byebye requests?

Technically it isn't necessary since the only useful information is the USN. But we want to stick with the GENA format that requires a NT header. In truth the requirement of including the NT header is a consequence of the next issue.

5.3.5. Shouldn't the NT and NTS values be switched?

Yes, they should. Commands such as ssdp:alive and ssdp:byebye should be NT values and the service type, where necessary, should be the NTS. The current mix-up is a consequence of a previous design where the NT header was used in a manner much like we use the USN today. This really needs to change.

6. SSDP Auto-Shut-Off Algorithm

6.1. Problem Statement

A mechanism is needed to ensure that SSDP does not cause such a high level of traffic that it overwhelms the network it is running on.

6.2. Proposed Solution

[Ed. Note: We have a proposed solution but it is still a bit rough, so we will be publishing to the SSDP mailing list for further discussion before including it in the draft.]

6.3. Design Rationale

6.3.1. Why do we need an auto-shut-off algorithm?

The general algorithm for figuring out how much bandwidth SSDP uses over a fixed period of time based on the number of ssdp:discover requests is :

DR = Total number of SSDP clients making ssdp:discover requests over the time period in question.
RS = Total number of services that will respond to the ssdp:discover requests over the time period in question.
AM = Average size of the ssdp:discover requests/responses.
TP = Time period in question.

((DR*3 + DR*9*RS)*AM)/TP

The 3 is the number of times the ssdp:discover request will be repeated.
The 9 is the number of times the unicast responses to the ssdp:discover requests will be sent out assuming the worst case in which all 3 original requests are received.

So let’s look at a real world worst-case scenario. Some companies, in order to enable multicast based services such as voice or video streaming to be easily configured set their local administrative multicast scope to encompass their entire company. This means one gets networks with 100,000 machines in a single administrative multicast scope. Now imagine that there is a power outage and all the machines are coming back up at the same time. Further imagine that they all want to refresh their printer location caches so they all send out ssdp:discover requests. Let us finally imagine that there are roughly 5000 printers in this network. To simplify the math we will assume that the ssdp:discover requests are evenly distributed over the 30 seconds.

DR = 100,000 requesting clients
RS = 5000 services
AM = 512 bytes
TP = 30 seconds

((100000*3+100000*9*5000)*512)/30 = 76805120000 bytes/s =
585976.5625 Megabits per second

This is what one would call an awful number.

ROKU EXH. 1002

In a more reasonably sized network SSDP is able to handle this worst case scenario much better. For example, let’s look at a network with 1000 clients and 50 printers.

```
DR = 1000 requesting clients
RS = 50 services
AM = 512 bytes
TP = 30 seconds
```

((1000*3+1000*9*50)*512)/30 = 7731200 bytes/s = 59 Mbps

Now this looks like an awful amount but remember that that this is the total data rate needed for 30 seconds. This means that the total amount of information SSDP needs to send out to survive a reboot is 59*30 = 1770 Mb. Therefore a 10 Mbps network, assuming an effective data rate 5 Mbps under constant load that means it will take 1770/5 = 354 seconds = 6 minutes for the network to settle down.

That isn’t bad considering that this is an absolute worst case in a network with 1000 clients and 50 services all of whom want to talk to each other at the exact same instant.

In either case, there are obvious worst-case scenarios and we need to avoid network storms, therefore we need a way for SSDP to de-activate before it causes a network storms.

6.3.2. Why not just require everyone to support directories and thus get around the scaling issue?

Many manufacturers stick every protocol they can think of in their clients and services. So if your network administrator happened to buy some clients and servers that supported SSDP but didn’t know they supported SSDP then you can imagine the problems. Therefore even if we required directory support there are still many cases where SSDP clients and services may inadvertently end up in a network without anyone knowing it and cause problems.

7. ssdp:all

7.1. Problem Statement

A mechanism is needed to enable a client to enumerate all the services available on a particular SSDP multicast channel/port.

7.2. Proposed Solution

All SSDP services MUST respond to SEARCH requests over the SSDP multicast channel/port with the ST value of ssdp:all by responding as if the ST value had been their service type.

For brevity’s sake a SEARCH request with a ST of ssdp:all will be referred to as a ssdp:all request.

7.3. Design Rationale

7.3.1. Why would anyone want to enumerate all services?

This feature is mostly for network analysis tools. It also will prove very useful in the feature when directories become SSDP aware. They will be able to discover all services, record information about them and make that information available outside the local administrative multicast scope.

8. SSDP Reserved Multicast Channel

8.1. Problem Statement

SSDP needs a local administrative multicast channel that will be guaranteed to only be used by SSDP compliant clients and services.

8.2. Proposed Solution

IANA has reserved the relative multicast address "5" for the exclusive use of SSDP. In the local administrative scope used by this version of SSDP the relative address translates to 239.255.255.250.

An application has been put in for a SSDP reserved port but IANA has not yet responded.

8.3. Design Rationale

8.3.1. Why didn't SSDP just get a static local administrative scope address rather than a relative address?

We got a relative address because we expect that SSDP may be used to discover basic system services such as directories. In that case if you can't find a directory in your local scope you may want to try a wider multicast scope. This is exactly the sort of functionality enabled by MALLOC (http://www.ietf.org/html.charters/malloc-charter.html). MALLOC allows one to enumerate all the multicast scopes that are supported on the network. The SSDP client can then try progressively larger scopes to find the service they are seeing. However this progressively wider discovery only works if SSDP uses a relative address.

8.3.2. Why does SSDP need to use a port other than 80?

There is a bug in the Berkley Sockets design that was inherited by WinSock as well. The bug is as follows: One can not grab a particular port on a particular multicast address without owning the same port on the local unicast address.

The result is that if we used port 80 on the SSDP multicast scope then we would require that the SSDP software also grab port 80 for the local machine. This would mean that SSDP could only be implemented on machines which either didn't have HTTP servers or whose HTTP servers had been enhanced to support SSDP.

We felt this was a unnecessary restriction. Therefore we are choosing to use a port other than 80 on the SSDP multicast channel.

9. HTTP Headers

9.1. USN Header

USN = "USN" ":" AbsoluteURI; defined in section 3.2.1 of [RFC2616]

9.2. ST Header

ST = "ST" ":" AbsoluteURI

10. Security Considerations

TBD.

11. IANA Considerations

To ensure correct interoperation based on this specification, IANA must reserve the URI namespace starting with "ssdp:" for use by this specification, its revisions, and related SSDP specifications.

IANA has reserved the relative multicast address "5" for exclusive use by SSDP. An application has been made for a registered port.

12. Appendix - Constants

MAX_UNIQUE - 50 - Maximum number of unique IP address/port pairs that may be sent over UDP before tripping the auto-shut-off algorithm.

MAX_COUNT - 30 seconds - When the "go quiet" process is begun a message is sent out that is delayed a random interval between 0 to MAX_COUNT seconds.

13. Acknowledgements

This document is the result of enormous effort by a large number of people including but not limited to:
Alan Boshier, Babak Jahromi, Brandon Watson, Craig White, Dave Thaler, Holly Knight, Michel Guittet, Mike Zintel, Munil Shah, Paul Moore, Peter Ford, Pradeep Bahl, and Todd Fisher.

14. References

[HTTPUDP] Y. Y. Goland. Multicast and Unicast UDP HTTP Requests. Internet Draft - a work in progress, draft-goland-http-udp-00.txt.

[GENA] J. Cohen, S. Aggarwal, Y. Y. Goland. General Event Notification Architecture Base: Client to Arbiter. Internet Draft - a work in progress, draft-cohen-gena-client-00.txt.

[MAN] H. Nielsen, P. Leach, S. Lawrence. Mandatory Extensions in HTTP. Internet Draft - a work in progress, draft-frystyk-http-extensions-03.txt.

[RFC2119] S. Bradner. Key words for use in RFCs to Indicate Requirement Levels. RFC 2119, March 1997.

[RFC2365] D. Meyer. Administratively Scoped IP Multicast. RFC 2365, July 1998.

[RFC2396] T. Berners-Lee, R. Fielding and L. Masinter. Uniform Resource Identifiers (URI): Generic Syntax. RFC 2396, August 1998.

[RFC2518] Y. Goland, E. Whitehead, A. Faizi, S. Carter, and D. Jensen. HTTP Extensions for Distributed Authoring Ã» WEBDAV. RFC 2518, February 1999.

[RFC2616] R. Fielding, J. Gettys, J. C. Mogul, H. Frystyk, L. Masinter, P. Leach and T. Berners-Lee. Hypertext Transfer Protocol - HTTP/1.1. RFC 2616, November 1998.

[DASL] S. Reddy, D. Lowry, S. Reddy, R. Henderson, J. Davis, A. Babich. DAV Searching & Locating. a work in progress - draft-ietf-dasl-protocol-00.txt.

15. Author's Addresses

Yaron Y. Goland, Ting Cai, Paul Leach, Ye Gu
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052

Email: {yarong, tingcai, paulle, yegu}@microsoft.com

Shivaun Albright
Hewlett-Packard Company
Roseville, CA

Email: SHIVAUN_ALBRIGHT@HP-Roseville-om2.om.hp.com

This document will expire in April 2000.

APPENDIX R

Network Working Group
Request for Comments: 2608
Updates: 2165
Category: Standards Track

E. Guttman
C. Perkins
Sun Microsystems
J. Veizades
@Home Network
M. Day
Vinca Corporation

June 1999

Service Location Protocol, Version 2

Status of This Memo

This document specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "Internet Official Protocol Standards" (STD 1) for the standardization state and status of this protocol. Distribution of this memo is unlimited.

Copyright Notice

Abstract

The Service Location Protocol provides a scalable framework for the discovery and selection of network services. Using this protocol, computers using the Internet need little or no static configuration of network services for network based applications. This is especially important as computers become more portable, and users less tolerant or able to fulfill the demands of network system administration.

Table of Contents

```
    1. Introduction                                                  3
        1.1. Applicability Statement  . . . . . . . . . . . . . . .   3
    2. Terminology                                                   4
        2.1. Notation Conventions . . . . . . . . . . . . . . . . .   4
    3. Protocol Overview                                             5
    4. URLs used with Service Location                               8
        4.1. Service: URLs  . . . . . . . . . . . . . . . . . . . .   9
        4.2. Naming Authorities   . . . . . . . . . . . . . . . . .  10
        4.3. URL Entries  . . . . . . . . . . . . . . . . . . . . .  10
    5. Service Attributes                                            10
    6. Required Features                                             12
        6.1. Use of Ports, UDP, and Multicast   . . . . . . . . . .  13
```

```
       6.2. Use of TCP   . . . . . . . . . . . . . . . . . . . . . .  14
       6.3. Retransmission of SLP messages   . . . . . . . . . . . .  15
       6.4. Strings in SLP messages  . . . . . . . . . . . . . . . .  16
             6.4.1. Scope Lists in SLP . . . . . . . . . . . . . . .  16
   7. Errors                                                          17
   8. Required SLP Messages                                           17
       8.1. Service Request  . . . . . . . . . . . . . . . . . . . .  19
       8.2. Service Reply  . . . . . . . . . . . . . . . . . . . . .  21
       8.3. Service Registration . . . . . . . . . . . . . . . . . .  22
       8.4. Service Acknowledgment . . . . . . . . . . . . . . . . .  23
       8.5. Directory Agent Advertisement. . . . . . . . . . . . . .  24
       8.6. Service Agent Advertisement. . . . . . . . . . . . . . .  25
   9. Optional Features                                               26
       9.1. Service Location Protocol Extensions . . . . . . . . . .  27
       9.2. Authentication Blocks  . . . . . . . . . . . . . . . . .  28
             9.2.1. SLP Message Authentication Rules . . . . . . . .  29
             9.2.2. DSA with SHA-1 in Authentication Blocks  . . . .  30
       9.3. Incremental Service Registration   . . . . . . . . . . .  30
       9.4. Tag Lists  . . . . . . . . . . . . . . . . . . . . . . .  31
  10. Optional SLP Messages                                           32
      10.1. Service Type Request   . . . . . . . . . . . . . . . . .  32
      10.2. Service Type Reply   . . . . . . . . . . . . . . . . . .  32
      10.3. Attribute Request  . . . . . . . . . . . . . . . . . . .  33
      10.4. Attribute Reply  . . . . . . . . . . . . . . . . . . . .  34
      10.5. Attribute Request/Reply Examples . . . . . . . . . . . .  34
      10.6. Service Deregistration   . . . . . . . . . . . . . . . .  36
  11. Scopes                                                          37
      11.1. Scope Rules  . . . . . . . . . . . . . . . . . . . . . .  37
      11.2. Administrative and User Selectable Scopes. . . . . . . .  38
  12. Directory Agents                                                38
      12.1. Directory Agent Rules  . . . . . . . . . . . . . . . . .  39
      12.2. Directory Agent Discovery  . . . . . . . . . . . . . . .  39
            12.2.1. Active DA Discovery  . . . . . . . . . . . . . .  40
            12.2.2. Passive DA Advertising . . . . . . . . . . . . .  40
      12.3. Reliable Unicast to DAs and SAs. . . . . . . . . . . . .  41
      12.4. DA Scope Configuration   . . . . . . . . . . . . . . . .  41
      12.5. DAs and Authentication Blocks. . . . . . . . . . . . . .  41
  13. Protocol Timing Defaults                                        42
  14. Optional Configuration                                          43
  15. IANA Considerations                                             44
  16. Internationalization Considerations                             45
  17. Security Considerations                                         46
   A. Appendix:  Changes to the Service Location Protocol from
                 v1 to v2                                             48
   B. Appendix:  Service Discovery by Type:  Minimal SLPv2 Features  48
   C. Appendix:  DAAdverts with arbitrary URLs                        49
   D. Appendix:  SLP Protocol Extensions                              50
       D.1. Required Attribute Missing Option  . . . . . . . . . . .  50
```

E. Acknowledgments 50
F. References 51
G. Authors' Addresses 53
H. Full Copyright Statement 54

1. Introduction

The Service Location Protocol (SLP) provides a flexible and scalable framework for providing hosts with access to information about the existence, location, and configuration of networked services. Traditionally, users have had to find services by knowing the name of a network host (a human readable text string) which is an alias for a network address. SLP eliminates the need for a user to know the name of a network host supporting a service. Rather, the user supplies the desired type of service and a set of attributes which describe the service. Based on that description, the Service Location Protocol resolves the network address of the service for the user.

SLP provides a dynamic configuration mechanism for applications in local area networks. Applications are modeled as clients that need to find servers attached to any of the available networks within an enterprise. For cases where there are many different clients and/or services available, the protocol is adapted to make use of nearby Directory Agents that offer a centralized repository for advertised services.

This document updates SLPv1 [RFC 2165], correcting protocol errors, adding some enhancements and removing some requirements. This specification has two parts. The first describes the required features of the protocol. The second describes the extended features of the protocol which are optional, and allow greater scalability.

1.1. Applicability Statement

SLP is intended to function within networks under cooperative administrative control. Such networks permit a policy to be implemented regarding security, multicast routing and organization of services and clients into groups which are not be feasible on the scale of the Internet as a whole.

SLP has been designed to serve enterprise networks with shared services, and it may not necessarily scale for wide-area service discovery throughout the global Internet, or in networks where there are hundreds of thousands of clients or tens of thousands of services.

2. Terminology

```
      User Agent (UA)
                A process working on the user's behalf to establish
                contact with some service.  The UA retrieves service
                information from the Service Agents or Directory Agents.

      Service Agent (SA) A process working on the behalf of one or more
                services to advertise the services.

      Directory Agent (DA) A process which collects service
                advertisements.  There can only be one DA present per
                given host.

      Service Type Each type of service has a unique Service Type
                string.

      Naming Authority The agency or group which catalogues given
                Service Types and Attributes.  The default Naming
                Authority is IANA.

      Scope A set of services, typically making up a logical
                administrative group.

      URL A Universal Resource Locator [8].
```

2.1. Notation Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [9].

```
      Syntax        Syntax for string based protocols follow the
                    conventions defined for ABNF [11].

      Strings       All strings are encoded using the UTF-8 [23]
                    transformation of the Unicode [6] character set and
                    are NOT null terminated when transmitted.  Strings
                    are preceded by a two byte length field.

      <string-list> A comma delimited list of strings with the
                    following syntax:

                       string-list = string / string ',' string-list
```

In format diagrams, any field ending with a \ indicates a variable length field, given by a prior length field in the protocol.

3. Protocol Overview

The Service Location Protocol supports a framework by which client applications are modeled as 'User Agents' and services are advertised by 'Service Agents.' A third entity, called a 'Directory Agent' provides scalability to the protocol.

The User Agent issues a 'Service Request' (SrvRqst) on behalf of the client application, specifying the characteristics of the service which the client requires. The User Agent will receive a Service Reply (SrvRply) specifying the location of all services in the network which satisfy the request.

The Service Location Protocol framework allows the User Agent to directly issue requests to Service Agents. In this case the request is multicast. Service Agents receiving a request for a service which they advertise unicast a reply containing the service's location.

```
   +------------+ ----Multicast SrvRqst----> +---------------+
   | User Agent |                            | Service Agent |
   +------------+ <----Unicast SrvRply------ +---------------+
```

In larger networks, one or more Directory Agents are used. The Directory Agent functions as a cache. Service Agents send register messages (SrvReg) containing all the services they advertise to Directory Agents and receive acknowledgements in reply (SrvAck). These advertisements must be refreshed with the Directory Agent or they expire. User Agents unicast requests to Directory Agents instead of Service Agents if any Directory Agents are known.

```
 +-------+ -Unicast SrvRqst-> +-----------+ <-Unicast SrvReg- +--------+
 | User  |                    | Directory |                   |Service |
 | Agent |                    |   Agent   |                   | Agent  |
 +-------+ <-Unicast SrvRply- +-----------+ -Unicast SrvAck-> +--------+
```

User and Service Agents discover Directory Agents two ways. First, they issue a multicast Service Request for the 'Directory Agent' service when they start up. Second, the Directory Agent sends an unsolicited advertisement infrequently, which the User and Service Agents listen for. In either case the Agents receive a DA Advertisement (DAAdvert).

```
     +---------------+ --Multicast SrvRqst-> +-----------+
     |    User or    | <--Unicast DAAdvert-- | Directory |
     | Service Agent |                       |   Agent   |
     +---------------+ <-Multicast DAAdvert- +-----------+
```

Services are grouped together using 'scopes'. These are strings which identify services which are administratively identified. A scope could indicate a location, administrative grouping, proximity in a network topology or some other category. Service Agents and Directory Agents are always assigned a scope string.

A User Agent is normally assigned a scope string (in which case the User Agent will only be able to discover that particular grouping of services). This allows a network administrator to 'provision' services to users. Alternatively, the User Agent may be configured with no scope at all. In that case, it will discover all available scopes and allow the client application to issue requests for any service available on the network.

```
+---------+   Multicast  +-----------+   Unicast   +-----------+
| Service | <--SrvRqst-- |   User    | --SrvRqst-> | Directory |
|  Agent  |              |   Agent   |             |   Agent   |
| Scope=X |   Unicast    | Scope=X,Y |   Unicast   |  Scope=Y  |
+---------+ --SrvRply--> +-----------+ <-SrvRply-- +-----------+
```

In the above illustration, the User Agent is configured with scopes X and Y. If a service is sought in scope X, the request is multicast. If it is sought in scope Y, the request is unicast to the DA. Finally, if the request is to be made in both scopes, the request must be both unicast and multicast.

Service Agents and User Agents may verify digital signatures provided with DAAdverts. User Agents and Directory Agents may verify service information registered by Service Agents. The keying material to use to verify digital signatures is identified using a SLP Security Parameter Index, or SLP SPI.

Every host configured to generate a digital signature includes the SLP SPI used to verify it in the Authentication Block it transmits. Every host which can verify a digital signature must be configured with keying material and other parameters corresponding with the SLP SPI such that it can perform verifying calculations.

SAs MUST accept multicast service requests and unicast service requests. SAs MAY accept other requests (Attribute and Service Type Requests). SAs MUST listen for multicast DA Advertisements.

The features described up to this point are required to implement. A minimum implementation consists of a User Agent, Service Agent or both.

There are several optional features in the protocol. Note that DAs MUST support all these message types, but DA support is itself optional to deploy on networks using SLP. UAs and SAs MAY support these message types. These operations are primarily for interactive use (browsing or selectively updating service registrations.) UAs and SAs either support them or not depending on the requirements and constraints of the environment where they will be used.

Service Type Request	A request for all types of service on the network. This allows generic service browsers to be built.
Service Type Reply	A reply to a Service Type Request.
Attribute Request	A request for attributes of a given type of service or attributes of a given service.
Attribute Reply	A reply to an Attribute Request.
Service Deregister	A request to deregister a service or some attributes of a service.
Service Update	A subsequent SrvRqst to an advertisement. This allows individual dynamic attributes to be updated.
SA Advertisement	In the absence of Directory Agents, a User agent may request Service Agents in order to discover their scope configuration. The User Agent may use these scopes in requests.

In the absence of Multicast support, Broadcast MAY be used. The location of DAs may be staticly configured, discovered using SLP as described above, or configured using DHCP. If a message is too large, it may be unicast using TCP.

A SLPv2 implementation SHOULD support SLPv1 [22]. This support includes:

1. SLPv2 DAs are deployed, phasing out SLPv1 DAs.

2. Unscoped SLPv1 requests are considered to be of DEFAULT scope. SLPv1 UAs MUST be reconfigured to have a scope if possible.

3. There is no way for an SLPv2 DA to behave as an unscoped SLPv1 DA. SLPv1 SAs MUST be reconfigured to have a scope if possible.

4. SLPv2 DAs answer SLPv1 requests with SLPv1 replies and SLPv2 requests with SLPv2 replies.

5. SLPv2 DAs use registrations from SLPv1 and SLPv2 in the same way. That is, incoming requests from agents using either version of the protocol will be matched against this common set of registered services.

6. SLPv2 registrations which use Language Tags which are greater than 2 characters long will be inaccessible to SLPv1 UAs.

7. SLPv2 DAs MUST return only service type strings in SrvTypeRply messages which conform to SLPv1 service type string syntax, ie. they MUST NOT return Service Type strings for abstract service types.

8. SLPv1 SrvRqsts and AttrRqsts by Service Type do not match Service URLs with abstract service types. They only match Service URLs with concrete service types.

SLPv1 UAs will not receive replies from SLPv2 SAs and SLPv2 UAs will not receive replies from SLPv1 SAs. In order to interoperate UAs and SAs of different versions require a SLPv2 DA to be present on the network which supports both protocols.

The use of abstract service types in SLPv2 presents a backward compatibility issue for SLPv1. It is possible that a SLPv1 UA will request a service type which is actually an abstract service type. Based on the rules above, the SLPv1 UA will never receive an abstract Service URL reply. For example, the service type 'service:x' in a SLPv1 AttrRqst will not return the attributes of 'service:x:y://orb'. If the request was made with SLPv2, it would return the attributes of this service.

4. URLs used with Service Location

A Service URL indicates the location of a service. This URL may be of the service: scheme [13] (reviewed in section 4.1), or any other URL scheme conforming to the URI standard [8], except that URLs without address specifications SHOULD NOT be advertised by SLP. The service type for an 'generic' URL is its scheme name. For example, the service type string for "http://www.srvloc.org" would be "http".

Reserved characters in URLs follow the rules in RFC 2396 [8].

4.1. Service: URLs

Service URL syntax and semantics are defined in [13]. Any network service may be encoded in a Service URL.

This section provides an introduction to Service URLs and an example showing a simple application of them, representing standard network services.

A Service URL may be of the form:

```
   "service:"<srvtype>"://"<addrspec>
```

The Service Type of this service: URL is defined to be the string up to (but not including) the final ':' before <addrspec>, the address specification.

<addrspec> is a hostname (which should be used if possible) or dotted decimal notation for a hostname, followed by an optional ':' and port number.

A service: scheme URL may be formed with any standard protocol name by concatenating "service:" and the reserved port [1] name. For example, "service:tftp://myhost" would indicate a tftp service. A tftp service on a nonstandard port could be "service:tftp://bad.glad.org:8080".

Service Types SHOULD be defined by a "Service Template" [13], which provides expected attributes, values and protocol behavior. An abstract service type (also described in [13]) has the form

```
   "service:<abstract-type>:<concrete-type>".
```

The service type string "service:<abstract-type>" matches all services of that abstract type. If the concrete type is included also, only these services match the request. For example: a SrvRqst or AttrRqst which specifies "service:printer" as the Service Type will match the URL service:printer:lpr://hostname and service:printer:http://hostname. If the requests specified "service:printer:http" they would match only the latter URL.

An optional substring MAY follow the last '.' character in the <srvtype> (or <abstract-type> in the case of an abstract service type URL). This substring is the Naming Authority, as described in Section 9.6. Service types with different Naming Authorities are quite distinct. In other words, service:x.one and service:x.two are different service types, as are service:abstract.one:y and service:abstract.two:y.

4.2. Naming Authorities

A Naming Authority MAY optionally be included as part of the Service Type string. The Naming Authority of a service defines the meaning of the Service Types and attributes registered with and provided by Service Location. The Naming Authority itself is typically a string which uniquely identifies an organization. IANA is the implied Naming Authority when no string is appended. "IANA" itself MUST NOT be included explicitly.

Naming Authorities may define Service Types which are experimental, proprietary or for private use. Using a Naming Authority, one may either simply ignore attributes upon registration or create a local-use only set of attributes for one’s site. The procedure to use is to create a ’unique’ Naming Authority string and then specify the Standard Attribute Definitions as described above. This Naming Authority will accompany registration and queries, as described in Sections 8.1 and 8.3. Service Types SHOULD be registered with IANA to allow for Internet-wide interoperability.

4.3. URL Entries

```
     0                   1                   2                   3
     0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |   Reserved    |          Lifetime             |   URL Length  |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |URL len, contd.|            URL (variable length)              \
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |# of URL auths |            Auth. blocks (if any)              \
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

SLP stores URLs in protocol elements called URL Entries, which associate a length, a lifetime, and possibly authentication information along with the URL. URL Entries, defined as shown above, are used in Service Replies and Service Registrations.

5. Service Attributes

A service advertisement is often accompanied by Service Attributes. These attributes are used by UAs in Service Requests to select appropriate services.

The allowable attributes which may be used are typically specified by a Service Template [13] for a particular service type. Services which are advertised according to a standard template MUST register all service attributes which the standard template requires. URLs with schemes other than "service:" MAY be registered with attributes.

Non-standard attribute names SHOULD begin with "x-", because no standard attribute name will ever have those initial characters.

An attribute list is a string encoding of the attributes of a service. The following ABNF [11] grammar defines attribute lists:

```
attr-list = attribute / attribute ‘,’ attr-list
attribute = ‘(’ attr-tag ‘=’ attr-val-list ‘)’ / attr-tag
attr-val-list = attr-val / attr-val ‘,’ attr-val-list
attr-tag = 1*safe-tag
attr-val = intval / strval / boolval / opaque
intval = [-]1*DIGIT
strval = 1*safe-val
boolval = "true" / "false"
opaque = "\FF" 1*escape-val
safe-val = ; Any character except reserved.
safe-tag = ; Any character except reserved, star and bad-tag.
reserved = ‘(’ / ‘)’ / ‘,’ / ‘\’ / ‘!’  / ‘<’ / ‘=’ / ‘>’ / ‘~’ / CTL
escape-val = ‘\’ HEXDIG HEXDIG
bad-tag = CR / LF / HTAB / ‘_’
 star = ‘*’
```

The <attr-list>, if present, MUST be scanned prior to evaluation for all occurrences of the escape character ‘\’. Reserved characters MUST be escaped (other characters MUST NOT be escaped). All escaped characters must be restored to their value before attempting string matching. For Opaque values, escaped characters are not converted - they are interpreted as bytes.

Boolean	Strings which have the form "true" or "false" can only take one value and may only be compared with ’=’. Booleans are case insensitive when compared.
Integer	Strings which take the form [-] 1*<digit> and fall in the range "-2147483648" to "2147483647" are considered to be Integers. These are compared using integer comparison.
String	All other Strings are matched using strict lexical ordering (see Section 6.4).
Opaque	Opaque values are sequences of bytes. These are distinguished from Strings since they begin with the sequence "\FF". This, unescaped, is an illegal UTF-8 encoding, indicating that what follows is a sequence of bytes expressed in escape notation which constitute the binary value. For example, a ’0’ byte is encoded "\FF\00".

A string which contains escaped values other than from the reserved set of characters is illegal. If such a string is included in an <attr-list>, <tag-list> or search filter, the SA or DA which receives it MUST return a PARSE_ERROR to the message.

A keyword has only an <attr-tag>, and no values. Attributes can have one or multiple values. All values are expressed as strings.

When values have been advertised by a SA or are registered in a DA, they can take on implicit typing rules for matching incoming requests.

Stored values must be consistent, i.e., x=4,true,sue,\ff\00\00 is disallowed. A DA or SA receiving such an <attr-list> MUST return an INVALID_REGISTRATION error.

6. Required Features

This section defines the minimal implementation requirements for SAs and UAs as well as their interaction with DAs. A DA is not required for SLP to function, but if it is present, the UA and SA MUST interact with it as defined below.

A minimal implementation may consist of either a UA or SA or both. The only required features of a UA are that it can issue SrvRqsts according to the rules below and interpret DAAdverts, SAAdverts and SrvRply messages. The UA MUST issue requests to DAs as they are discovered. An SA MUST reply to appropriate SrvRqsts with SrvRply or SAAdvert messages. The SA MUST also register with DAs as they are discovered.

UAs perform discovery by issuing Service Request messages. SrvRqst messages are issued, using UDP, following these prioritized rules:

1. A UA issues a request to a DA which it has been configured with by DHCP.

2. A UA issues requests to DAs which it has been statically configured with.

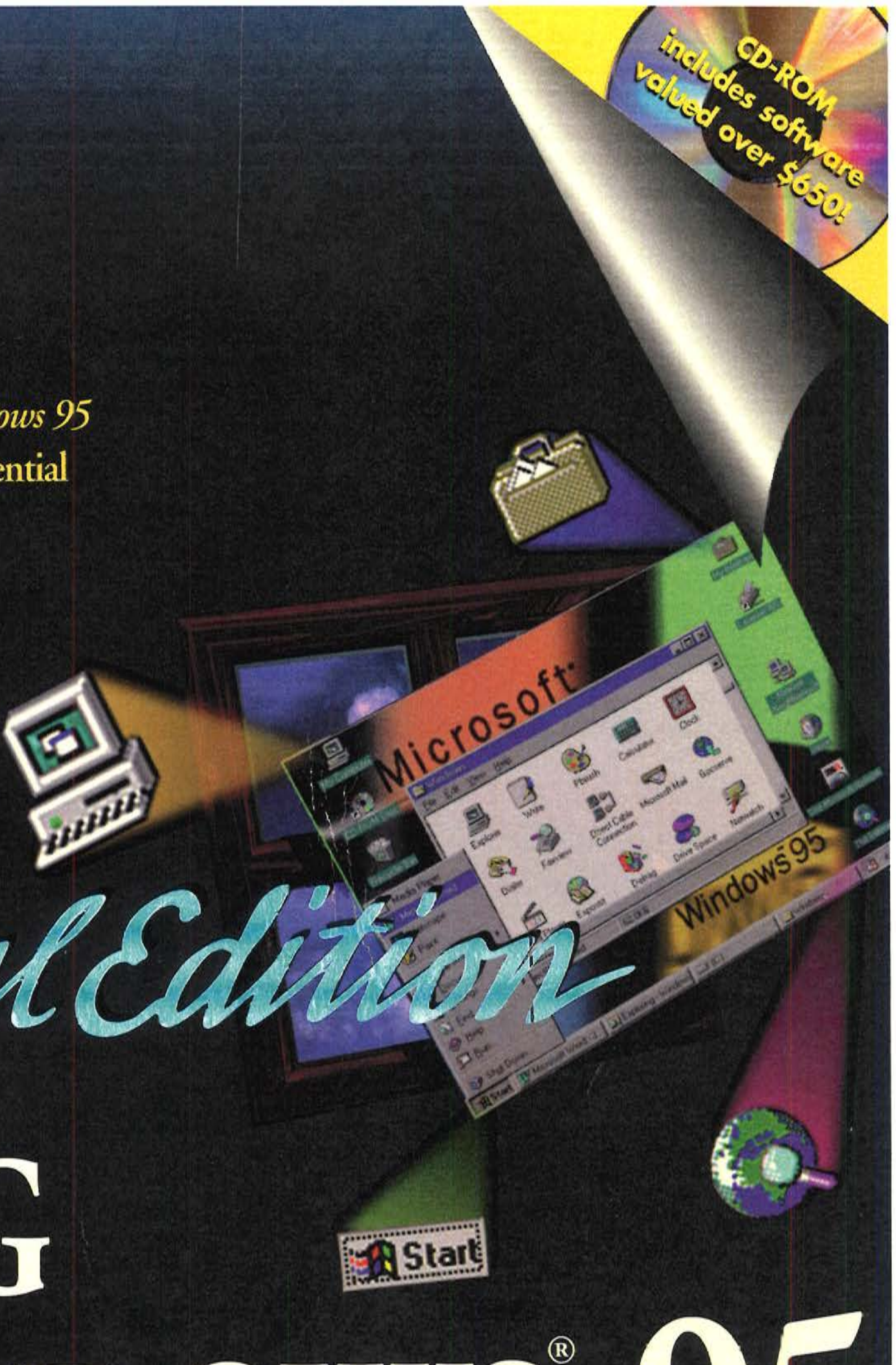
3. UA uses multicast/convergence SrvRqsts to discover DAs, then uses that set of DAs. A UA that does not know of any DAs SHOULD retry DA discovery, increasing the waiting interval between subsequent attempts exponentially (doubling the wait interval each time.) The recommended minimum waiting interval is CONFIG_DA_FIND seconds.

APPENDIX S

The Most Complete Reference

"*Special Edition Using Windows 95* will help you unlock the potential of Windows 95."

Jeffrey Sloman
Contributing Editor
Windows Magazine



Special Edition Using Windows® 95

Ron Person

que®

Special Edition

USING Windows® 95

Written by

Ron Person
R. Michael O'Mara
Gerald Paul Honeycutt Jr.
Roger Jennings
Rob Tidrow
Ian Stokell
Dick Cravens
William S. Holderby
Michael Marchuk
Gordon Meltzer
Glenn Fincher
Francis Moss
Paul E. Robichaux
Doug Kilarski
Jim Boyce
Sue Plumley
Alex Leavens
Lisa A. Bucki
Dave Plotkin
Peter Kent

que®

Special Edition Using Windows 95

Library of Congress Catalog No.: 95-70648

ISBN: 1-56529-921-3

97 96 95 6 5 4 3

Interpretation of the printing code: the rightmost double-digit number is the year of the book's printing; the rightmost single-digit number, the number of the book's printing. For example, a printing code of 95-1 shows that the first printing of the book occurred in 1995.

Screen reproductions in this book were created using Collage Plus from Inner Media, Inc., Hollis, NH.

Composed in *Stone Serif* and *MCPdigital* by Que Corporation.

Chapter 25

Working with Network Printers

by William S. Holderby

In Chapter 6, "Controlling Printers," you learned the basics of installing and working with printers—or at least those attached directly to your computer. Of course, not all printers are connected exclusively to your PC. In many workplaces, a local area network has multiple printer connections. Although Windows 95 makes network printers appear to operate as local printers, network printing may seem more complex. Local printers usually remain attached to the same port and are under your control. Network printers can change location and are controlled by other users or a network administrator. If problems arise when you are using a network printer, troubleshooting is much easier if you understand some of the differences between local and network printing.

This chapter takes printing a step further and discusses printing issues from a network perspective. Specifically, you learn how to:

- Print to network printers
- Optimize print resources
- Manage print files
- Solve common network printing problems
- Use custom Printer Drivers and utilities

Examining Windows 95's New Network Printing Features

Windows 95 incorporates several new features and enhancements that markedly improve network printing. These new features include the following:

- *Network Point and Print* enables users to copy printer drivers automatically from network print servers to their local PC. This reduces the time it takes to set up a new printer and eliminates the need to find and copy vendor driver software. This feature also eliminates the chance of configuring the wrong printer. You can access Network Point and Print from network servers running Windows 95, Windows NT Advanced Server, Windows NT Workstation, Windows for Workgroups 3.11, or Novell NetWare.
- *Windows 95's Network Neighborhood* provides tools to configure print resources quickly on Windows 95, Windows NT, and Novell servers. You can use this feature to find, use, and manage print jobs on printers interfacing any of these devices. Formerly, the user had to memorize locations and complex network commands. Network Neighborhood virtually eliminates this need through its new network user interface.
- *Compatibility with NetWare's PSERVER* enables you to access print jobs from NetWare's print spooler.
- *Deferred printing* provides you with the ability to save printouts until you reattach your printer. Deferred printing automatically stores print jobs after you detach your PC from the network, and automatically restarts them after you reestablish the connection.
- *Printer Driver* provides command resources to remotely stop, hold, cancel, or restart print jobs located on shared printers.

Understanding Network Printing

Before delving too deeply into the nuts and bolts of network printing, you first must become familiar with the terminology you will see frequently in this discussion:

- *LAN Administrators* provide a management function to the local area network by assisting users and directing what resources are available on the network.

- *Systems policies* are software controls that are created by LAN Administrators to define what users can and cannot do on their desktops and the network. For example, you might use a system policy to restrict access to certain network programs.
- A *client* is a workstation that uses the services of any network server that can include server-based software systems, printers, and mass storage devices.
- *Print queues* contain print jobs that are not immediately printed. A queue holds the job until the printer is ready to print.
- *Windows Redirector* is the software module contained in the Windows network architecture that identifies software references to network devices and connects those devices to the workstation through the network.
- *Network resources* are software and hardware features that are available from servers and other workstations on the LAN. Resources such as shared drivers and server-based programs are available for network users.
- *Printing resources* are LAN resources that are dedicated to serving network users for the purpose of printing. These include shared printers, network printers, and print queues.
- *Print servers* service the printing needs of network clients.

Three network printer types are found on most networks:

- Printers attached to the network through a Microsoft Network compatible server.
- Printers connected to a server running a compatible network operating system other than Windows, such as Novell NetWare and Banyan VINES.
- Printers directly attached to a network through a special printer network interface card (NIC).

Printing from Applications

Printing to network printers from within applications requires the same commands and menu items that you use to print locally. Windows handles the network communications and creates a Printer Driver for each attached network printer. As with local printers, you can access network printer configuration information in the Printer Properties sheet. In this sheet, you can change the network printer's properties for default or specific printing tasks.

◀ See "Options for Your Printer," p. 187

Caution

Remember that other users can change a network printer configuration. Before printing, check all important print settings for this printer on your PC, including paper orientation and resolution. Don't assume that they are already set the way that you want them. Printing mistakes on network printers take extra time to recover.

When applications create a print file, they send a print stream to the network server through the Windows 95 Network Redirector. A print file contains spooled printer data and commands that are being temporarily stored prior to printing. The Network Redirector, which is part of Windows 95 network architecture, determines whether the print stream destination is a local printer. A print stream is the data that is being sent to a printer containing both printable and unprintable characters. Unprintable characters are used to control the printer. It also uses Windows network drivers to locate the designated printer.

Drag-and-Drop Printing

◀ See "Drag-and-Drop Printing from the Desktop," p. 178

To perform drag-and-drop printing, you use the same procedure as you do for local printing. Remember, however, that drag-and-drop printing sends the print job to the system's default printer. If the selected printer is not the default printer, Windows will ask you to make it the default printer prior to printing the file. When initially connecting your PC to a network, this printer might not be available. Be sure to log in to the network and verify the printer's network connection before setting it as the default printer (unless you plan to use deferred printing with your default printer in which case anything you print is saved by Windows until the printer becomes available).

Installing a Network Printer

Network printers are usually installed in one of two ways:

- The *Add Printer Wizard* from the Printers folder can be used for any printer connected to the network. The installation of a network printer doesn't change the printer, it simply loads an appropriate printer driver on your PC. Windows 95 uses that driver during printing.

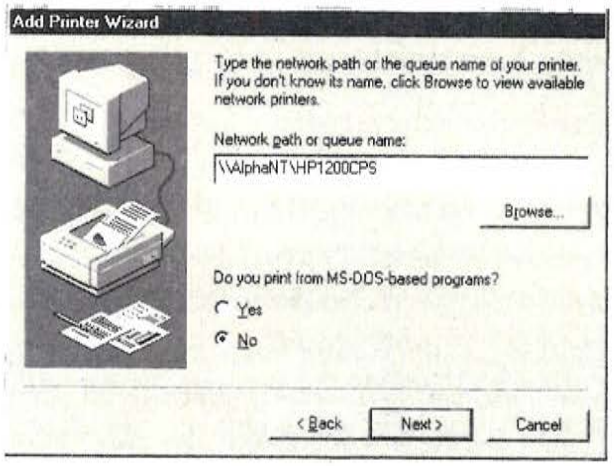
- *Point and Print installation* from the desktop can be used for printers attached to servers that are Microsoft Client compatible.

Using the Add Printer Wizard

Installing a network printer involves the same Add Printer Wizard as the local printer installation described in Chapter 6, "Controlling Printers." However, there are some differences.

When you configure a local printer, the location of your cable to a specific printer port determines the port's selection. The network printer, on the other hand, requires a network resource name. In the example shown in figure 25.1, an HP 1200 CPS print named HP1200CPS is located on the AlphaNT server.

◄ See "Connecting Windows 95 to a Novell Network," p. 673



Type the network path or the queue name of your printer. If you don't know its name, click Browse to view available network printers.

Network path or queue name: \\AlphaNT\HP1200CPS

Do you print from MS-DOS-based programs? Yes [✓ No]

Fig. 25.1 The Add Printer Wizard requires a network address for printer installation.

If you're not sure of the correct address for the network printer, you can choose to browse the network. Browsing enables you to check which network printers are currently available. Some servers require passwords to view what network resources they have available. If you desire access to a server, but do not know the password, contact your LAN Administrator.

◄ See "Setting Up a Windows 95 Peer-to-Peer Network," p. 617

To configure a network printer, you need to know its make and model. You can get this information from your network administrator. Microsoft network servers enable you to install printer drivers quickly.

To set up a network printer with the Add Printer Wizard, follow these steps:

1. Choose Start, Settings, Printers, then double-click on the Add Printer folder.
2. From the first Add Printer Wizard screen, click the Next button. Windows 95 then displays the next Wizard screen, which asks you to decide if you are adding a Network or a Local Printer.
3. Choose the Network Printer option to connect your PC to a network printer. Choose the Next button located at the bottom of the window.
4. Next, you must identify the network path to the printer (refer to fig. 25.1). Select the Browse button to view the Network Neighborhood.
5. The Network Neighborhood displays a list of all servers and workstations connected to your network. Find the appropriate printer and select it. Then choose the Next button.

   The Wizard accesses the selected printer and determines whether its server can download an appropriate printer driver. If a driver is available, the Wizard automatically loads the driver and sets a default configuration for the printer. If a driver is not available, the Wizard asks you to specify the printer's make and model.

Tip
Use the Add Printer Wizard again if you have difficulty connecting to a printer using the Point and Print procedure.

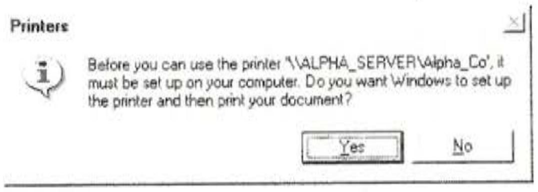
6. Select the manufacturer and printer model by scrolling the Wizard screen lists; then click Next. The screen now offers a default name for your printer. The name should adequately describe the printer for later identification.
7. The Wizard asks whether you want this printer to be your default printer; select Yes or No. Follow this decision by selecting the Next control.
8. The final wizard screen provides the controls to print a test page on the printer you just installed. You can print the test page by selecting Yes; or select No to not print the test page. As a general rule, you should always print a test page to verify the successful completion of the Add a Printer Wizard.
9. Click Finish.

Using Point and Print

Point and Print enables a workstation user to quickly connect to and use a printer shared on another Windows 95 workstation, a Windows NT

Advanced Server, or a Novell NetWare server. When first connecting to the shared printer, Windows 95 automatically copies and installs the correct driver for the shared printer from the server.

1. Choose the Network Neighborhood icon on the Windows desktop.
2. Choose the Entire Network icon. Windows displays all of the servers attached to your network.
3. Choose the Server that supports the printer you want to attach to your workstation. If you don't know which Server that is, ask the LAN Administrator or select each server in sequence until you find the name of the appropriate printer or print queue. Windows displays the server's screen showing its shared resources.
4. Drag a network printer icon from a server's window and drop it on the desktop. You receive a diagnostic message that says `You cannot move or copy this item to this location. Do you want to create a shortcut to the item instead?` Answer Yes. Windows creates a shortcut icon and drops it on the desktop.
5. Drag a document from a local folder and drop it on the New Printer folder icon. Windows displays an information screen such as that shown in figure 25.2. If you select Yes, Windows automatically connects to the printer and downloads the appropriate printer driver from the network printer's server. After loading and configuring the driver, Windows 95 begins printing to the network printer.


Printers

Before you can use the printer '\\ALPHA_SERVER\Alpha_Co', it must be set up on your computer. Do you want Windows to set up the printer and then print your document?

Fig. 25.2
If the printer driver is not loaded when you use Point and Print, Windows lets you install the driver on the fly.

Printing on a NetWare Network

To use a NetWare print queue, you must be logged onto a NetWare server. Windows 95 utilizes a PSERVER that can redirect print jobs from NetWare print queues to printers connected to Windows 95 workstations. In addition to the PSERVER capability, your PC must also use the Microsoft Client for NetWare Networks. Windows 95 automatically adapts to NetWare's security for printer and print queue access.

Note

The network administrator can control how NetWare shares printers. If the administrator uses a *policy file* to disable print sharing, the network cannot access the printers. A policy file contains a set of commands that are used by your network administrator to set rules for the operation and configuration of Windows on a network. When entering a new network, check with your network administrator for a sharing policy before attempting to configure shared printing.

NetWare Print Servers

Windows 95 provides printer services for NetWare networks, including a 32-bit PSERVER capability. PSERVER connects NetWare queues to printers shared by Windows 95 PCs. A NetWare print queue contains all the jobs waiting (queued) for a specific printer.

To connect your workstation to a Novell NetWare print server, follow these steps:

1. Choose the Network Neighborhood icon on the desktop. Notice that all servers (Microsoft and NetWare) appear in the Network Neighborhood screen. This screen displays an icon for each network drive currently attached to your system.

2. If you want to attach your workstation to a NetWare printer, choose the appropriate server by double-clicking its icon. The server dialog box opens and displays the shared directories, files, and print queues that are attached to the selected server.

3. Select the appropriate print queue. Choose File, Print and then select the Capture Printer Port control button.

4. The Capture Printer Port dialog box contains the names of currently unattached LPT ports. Select the Reconnect at Logon check box if you want to maintain this connection and have it attached when you re-start Windows. Then click OK to attach this print queue to your PC.

   Capturing the printer port attaches the NetWare print queue to the specified port. It does not, however, attach the associated printer to the desktop.

5. To attach the printer associated with that print queue, choose File, Create Shortcut. Windows displays a message warning that you cannot

ROKU EXH. 1002

configure a shortcut printer icon in the Create Shortcut dialog box, but that you can create the icon on the desktop. Click Yes to create the icon.

6. Double-click the printer icon. Windows asks whether you want to set up a printer. Choose Yes.

7. Windows then displays the Add Printer Wizard. Follow the Wizard to finish installing an appropriate printer for the desktop.

Note

You must know the type of printer attached to this print queue. This procedure is different than installing a printer attached to a Microsoft print server.

After configuring the printer, you can print by using the Point and Print procedure on the desktop.

Microsoft Client for NetWare

Windows 95 Microsoft Client for NetWare Networks enables you to connect to new or existing NetWare servers to send files and interact with server-based software. With Microsoft Client for NetWare Networks, you can browse and queue your print jobs using either the Windows 95 network user interface or existing Novell NetWare utilities. The Microsoft Client for NetWare interfaces work equally well with both NetWare 3.x and 4.x servers.

To use Microsoft Client for NetWare Networks, follow these steps:

1. Choose File, Print.

2. Your application might ask you to choose a destination printer. Most applications display a list of attached printers from which you can choose. If so, choose an appropriate network printer. Then choose OK.

   The Windows Redirector accepts the print stream and sends it to the selected printer over the network. Information concerning the status of the printing process automatically returns to you.

3. To monitor the status of your print job on a network printer, open the printer's folder and double-click the appropriate icon. The printer's local Printer Driver opens a status dialog box listing all print jobs in the printer's queue.

Point and Print for NetWare Print Queues

You can enable a Point and Print procedure to use a NetWare-compatible client as a destination. To do so, use the Point and Print procedure discussed earlier.

To print from the desktop to a network printer, follow these steps:

1. Open the folder that contains the document you want to print.
2. Select a document. Hold down the left mouse button and drag the selected document to the network printer's icon on your desktop. The document now appears as an outline.
3. Release the outlined document icon over the printer's icon. Windows 95 interprets the file type, starts the application associated with the file, commands the application to print the document, redirects the print job to the selected printer, and shuts down the application when the print job finishes.

Note

Before Windows can perform the desktop printing operation, you must associate the document with an installed application. If the document is not associated with such an application, Windows displays a message box informing you that it cannot perform the printing task.

Printing on a Microsoft Network

To share files and printers on Microsoft networks, you also can set user rights remotely through the User Manager in a Windows NT Advanced Server.

To connect to a Microsoft print server, follow these steps:

1. Click the Network Neighborhood icon on the desktop. Notice that all servers (Microsoft and NetWare) appear on the Network Neighborhood window. This screen displays an icon for each network drive currently attached to your system.
2. To attach a printer through a Microsoft server, choose the appropriate Microsoft server by double-clicking that server's icon. The server dialog box, named after the appropriate server, appears and displays the shared directories, files, and printers attached to the Microsoft server.

3. Select the appropriate printer queue. Choose File, Print and then select the capture printer port. The Capture Printer Port dialog box is displayed containing the name of a currently unattached LPT port.

4. In the Capture Printer Port dialog box, select the Reconnect at Logon check box if you want to maintain this connection and have it attached when you restart Windows. Then click the OK button to attach this print queue to your PC.

Note

Capturing the printer port attaches the printer to a specified port, but does not attach the associated printer to the desktop.

5. To attach the printer associated with the selected print queue, you must choose File, Create Shortcut. Windows displays a diagnostics message warning that you cannot configure a shortcut printer icon in the Create Shortcut dialog box, although Windows creates the shortcut on the desktop. Click Yes to create the icon.

6. Double-click the printer's shortcut icon. Windows asks whether you want to set up a printer. Choose Yes. Windows then displays the Add Printer Wizard.

7. Follow the Wizard to finish installing an appropriate printer for the desktop. The Add Printer Wizard identifies which printer make and model you are installing and completes the printer connection quickly.

After configuring the printer, you can print by using the Point and Print procedure on the desktop.

Troubleshooting

I can see a network printer using Network Neighborhood, but I can't print to it.

Try the following:

- Check with your network administrator about your access rights to the printer.
- Verify that the printer is properly configured on your PC.
- Check with other users to determine whether they can access the printer.
- Try to print to another printer on the network to check your network connectivity.

(continues)

(continued)

I can't stop, cancel, or delete a print job in a network queue.

Try the following:

- Check whether you have proper authorization from the network administrator to change the print settings. You might be authorized to change only your own print jobs, not others.
- If the print queue is on a shared printer, reload the printer driver or reset the printer properties. The system might not recognize that this printer is attached to your PC.

The network printer doesn't tell me that it is out of paper or toner.

Have the network administrator configure Winpopup to broadcast printer-problem announcements. Winpopup is a utility that comes with Windows. This utility allows the network and network users to send "popup" messages that identify events and get the attention of other network users.

Optimizing Print Resources

Network printing involves many of the same facilities as local printing. Applications create print files that the Network Redirector streams to the destination network printer. When working with network resources, however, you must consider several other issues to ensure that you're getting the most from your system.

Network Printer Configuration and Management

The Printer Properties sheet contains information on each local and network printer attached to your PC. Each printer's properties are specific to its make, model, and hardware configuration.

◀ See "Options for Your Printer," p. 187

You can make several changes to the properties to enhance your printing. The print quality can be enhanced by specific printers, setting device options, graphics, and the procedures for handling TrueType fonts. The following general procedure explores some of these changes:

1. After attaching a network printer, open its Printer Properties sheet by right-clicking the appropriate printer. From the pull-down menu that appears, choose Properties and then click the General tab. The pages in the Properties sheet are specific to your printer and display the options

and selections that match the printer's current hardware and print driver configuration.

2. Click the Device Options tab. Notice the options that the network printer offers.

3. Change the Device Options settings to match your printer's specifications. These options include such pertinent information as printer memory size and page protection. (If you don't see these options, check with your local area network [LAN] administrator.)

4. Click the Details tab. Check the spool settings to determine whether the printer is set to print after the first or last page spools. Usually, waiting until after the last page spools yields better results. Experiment with this setting to gain a better understanding of your configuration.

5. Click the Graphics tab. Change the dithering settings to identify which setting yields the best results for both speed and printout quality.

Troubleshooting

When I print to a printer on the network, my printout quality and settings are not consistent.

Try the following:

- Before printing, check with the printer's Properties sheet. Change the settings if required.
- Check with the system administrator for the printer settings, features, and hardware configuration. The printer might not be capable of handling your print job.
- Relate printout quality to changes in the property settings. Change your printer's properties and make test printouts to see how these changes affect the printouts.

Network Printer Drivers

Windows uses printer drivers to deliver your print files through the network to your printer. How well Windows performs this printing depends on how well the drivers perform. If you use drivers that are several revisions old, you might experience a slowdown. It is a good policy to check your printer drivers and update as revisions become available.

1. In the Control Panel, choose the System icon. Then click the Device Manager tab.
2. Verify that the network interface card driver is a virtual mode driver with a VxD extension. The driver will be listed including its extension. If a real-mode driver with a DRV extension is installed, then contact your LAN administrator or printer manufacturer for an updated revision.
3. Verify that the configured printer driver is a virtual mode driver. The driver should also have a VxD extension. If a real-mode driver with a DRV extension is installed, then contact your LAN administrator or printer manufacturer for an updated revision.
4. Ask your LAN administrator whether your system is configured with the latest driver version for your network printers. If the drivers are not the most current revision, request the latest update from either your LAN manager or the printer's manufacturer.

Managing Print Files and Sharing

After creating print files and sending them to a network printer, you must verify that the print jobs are finished, on hold, or need to be purged. You can check the print job status on both local and remote printers by using the Windows Printer Driver. Print job control is a complex task that involves user security rights on remote printers.

Viewing a Network Print Queue

Although you can view queue information, you cannot change any print job characteristics unless the LAN administrator has authorized you to do so. For some systems, the network administrator is the only user who can control all print jobs, while another user can control only his or her local shared-printing resources. LAN administrator policies determine which users can delete, pause, or purge documents from the queue. Usually, users can change the status of their individual print jobs, but not those of other users.

To view the queue, simply double-click on the printer's icon in the Printers folder or on the desktop. Windows displays the Printer Driver and print queue.

Shared Printing

Shared printing or *peer-to-peer sharing* provides other network workstations access to your local printer. Shared printing access is useful for transferring documents between workstations and for sharing expensive resources with other users. It is also an excellent way to maximize the use of often expensive printing hardware.

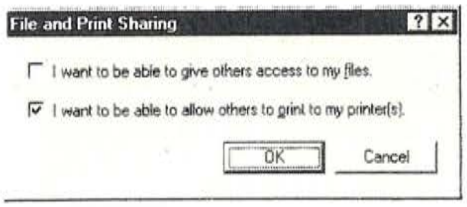
To share a printer, follow these steps:

1. Choose Start, Settings, Control Panel. In the Control Panel folder, double-click the Network icon. The Network tabs will appear. These tabs include Configuration, Identification, and Access Control.
2. On the Configuration page, select the Add button. The Select Network Component Type dialog box appears.
3. Choose Service and then click the Add button.
4. Choose Microsoft from the Manufacturers list box.
5. If your primary network logon client is Microsoft Networks, choose File and Printer Sharing for Microsoft Networks. If your primary network logon client is NetWare, choose File and Printer Sharing for NetWare Networks.
6. Choose OK to close the Select Network Service dialog box. For these changes to take effect, you must restart the computer.

Enabling Shared Printing

After configuring the network setup by following the preceding steps, you must enable the sharing feature as follows:

1. From the taskbar, choose Start, Settings, Control Panel. In the Control Panel folder, double-click the Network icon.
2. In the Network dialog box, choose the File and Print Sharing button.
3. In the File and Print Sharing dialog box, select the I Want to Be Able to Allow Others to Print to My Printer(s) check box (see fig. 25.3).


File and Print Sharing

☐ I want to be able to give others access to my files.

☑ I want to be able to allow others to print to my printer(s).

Fig. 25.3
The File and Print Sharing dialog box contains check boxes that enable you to share files and printers with other network users.

ROKU EXH. 1002

4. Choose OK to close the dialog box, and again to close the Network Control Panel. You must restart the computer for these changes to take effect.

Note

If the I Want to Be Able to Allow Others to Print to My Printer(s) check box is grayed (disabled), your system does not support print sharing.

Troubleshooting

My shared printer is unavailable to other workstations on my network.

Try the following:

- In the Control Panel, double-click the Network icon. Choose the File and Print Sharing button. In the File and Print Sharing dialog box, verify that the I Want to Be Able to Allow Others to Print to My Printer(s) check box is selected.
- Verify that all users are running a compatible protocol.
- Verify that your PC shows up in the network browser on other connected PCs.
- Verify that you can print successfully to your attached printer.
- Use the Extended Printer Troubleshooting (EPTS) application available in your Help file.

Disabling Shared Printing

After your workstation printer is shared, you might find that too many users are creating an overload. To disable the share, follow this procedure:

1. From the taskbar, choose Start, Settings, Control Panel. In the Control Panel folder, double-click the Network icon.
2. In the Network dialog box, choose the File and Print Sharing button.
3. Deselect the I Want to Be Able to Allow Others to Print to My Printer(s) check box.
4. Choose OK to close the dialog box, and again to close the Network Control Panel.

Creating Shared-Printer Security

In Windows 95, creating shared-resource security is a multistep procedure. In order to effectively share a resource, you must be able to control who accesses that resource and, to some extent, what they do with it. If you share your printer, you can impose some level of security. Securing your printer requires several steps.

1. Choose the Passwords icon in the Control Panel folder. The Password Properties sheet appears. This Properties sheet has three tabs: Passwords, Remote Administration, and User Profiles. (The Remote Administration tab is not present if file and printer sharing are not installed.)
2. Choose the Enable Remote Administration check box on the Remote Administration page.
3. Type a user-access password in the Passwords text box.
4. In the Confirm Passwords text box, retype the password. Record the password in your system workbook or manual.
5. Select OK.

Network users can now gain access to your system by using the password that you have just created. To access your shared printer, however, users must have the appropriate password information.

Deleting Connections to a Shared Printer

When you delete a shared connection between your workstation and a workstation sharing its local printer, disabling sharing keeps your local printer from being shared by the network.

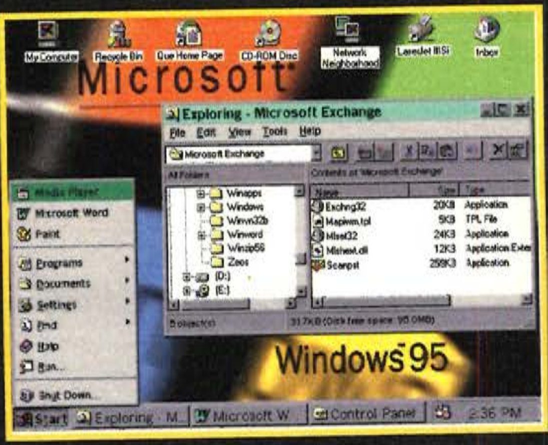
1. From the taskbar, choose Start, Settings, Printer's Folder. Windows displays a list of all printers, local or network, attached to your workstation.
2. Select the shared printer you want to delete.
3. Choose File, Delete.
4. Windows displays a dialog box warning that it will delete the selected printer. Click Yes.
5. Windows next displays a dialog box asking whether you want to delete this printer's drivers. Click Yes to delete the drivers.

Special Edition

Using Windows® 95

The secrets to mastering Windows 95!

See inside back cover for more information on the CD-ROM!



Que's *Special Edition Using Windows 95* is your comprehensive reference to maximizing Windows performance. From learning Windows navigation skills to advanced data-sharing techniques and tips on remote access, multimedia, OLE 2.0, and networking, this book is your complete guide to Windows 95.

Expert authors show you everything you need to know to rule Windows 95. You'll learn everything about the new user interface, file management, built-in utilities, and networking. You'll find out how to get online with The Microsoft Network and get connected to the Internet. And you'll learn how to put Windows 95 multimedia and numerous accessories to work for you.

- Master the new Windows 95 interface
- Use Windows' new tools including Explorer and Network Neighborhood
- Customize Windows 95 to fit your needs
- Quickly set up and use old and new applications
- Efficiently configure Windows 95 with Wizards
- Use Plug and Play to easily install new hardware
- Get the most from your laptop or notebook PC
- Build compound documents with OLE 2.0
- Connect to NetWare® and Microsoft® networks
- Get online with The Microsoft Network and the Internet
- Direct e-mail with Exchange
- Maximize the power of Windows 95 multimedia
- Get a hands-on approa[illegible] many real-world examp[illegible] tips, and troubleshooti[illegible]

Master the revolutionary new 32-bit operating system with Special Edition Using Windows 95 *from Que!*

User Level

New Casual Accomplished Expert

Category: Operating Systems
Covers: Version Windows 95

$39.99 USA / $53.99 CAN / £37.[illegible]

ISBN 1-56529-921-3
90000
0 29236 99213 2
9 781565 299214

QUE®

APPENDIX T

Mac OS 9:

THE MISSING MANUAL
The book that should have been in the box
ALL THE BASICS —AND BEYOND

David Pogue

POGUE PRESS™
O'REILLY®

ROKU EXH. 1002

Mac OS 9:

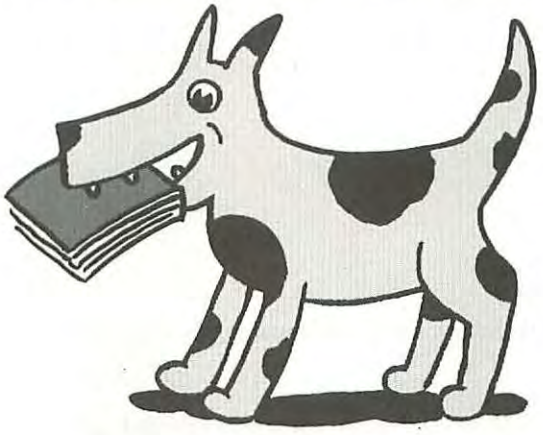
THE MISSING MANUAL

The book that should have been in the box

ROKU EXH. 1002

Mac OS 9:

THE MISSING MANUAL



David Pogue

POGUE PRESS™

O'REILLY®

Beijing • Cambridge • Farnham • Köln • Paris • Sebastopol • Taipei • Tokyo

Mac OS 9: The Missing Manual
by David Pogue

Published by Pogue Press/O'Reilly & Associates, Inc., 101 Morris Street, Sebastopol, CA 95472.

March 2000: First Edition.

This book is printed on acid-free paper with 85% recycled content, 15% post-consumer waste. O'Reilly & Associates is committed to using paper with the highest recycled content available consistent with high quality.

ISBN: 1-56592-857-1 [03/00]

CHAPTER
19

Printing, Fonts, and ColorSync

When Apple advertises the 50 new features of Mac OS 9, it could very well be talking just about the many new printing options, which, for the most part, appeal primarily to printing shops. After all, the printing industry is a Mac stronghold, so Apple's engineers spend a good amount of their energy making sure that every conceivable printing and font-management feature is built into the Mac.

Another elaborate set of software components is dedicated to the Mac OS's *desktop printing* feature, which is primarily useful to people who work on a network that offers several connected printers.

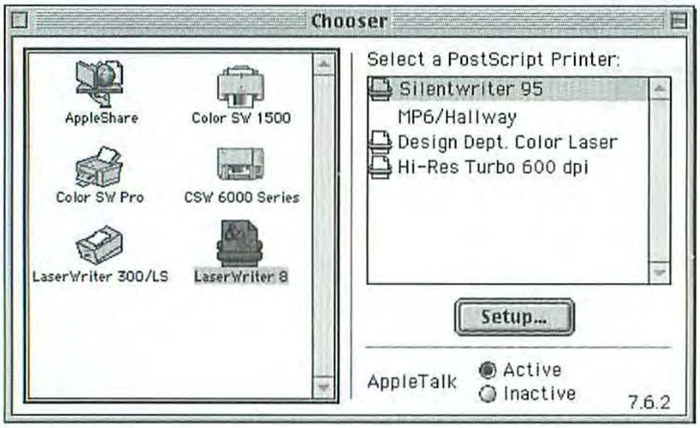
A final batch of features is aimed at people who scan, edit, and then print color photographs. As anyone who's tried to do such a thing knows, what's burnt umber to your scanner is rarely the same shade as burnt umber to your inkjet printer. Apple's ambitious ColorSync software is designed to be the negotiator between your scanner or camera, monitor, and printer, so that colors remain consistent throughout the process.

This chapter tackles all of these features: printing, managing multiple printers, fonts, and ColorSync.

Mac Meets Printer

Grand Central Station for Macintosh printers is the Chooser program (choose →Chooser). As you can see from Figure 19-1, it displays an icon for several discontinued Apple printers. (Apple no longer makes printers.) These icons represent the software—the *drivers*—that the Mac needs to communicate with each of the corresponding printers.

When you buy a modern printer from, say, Epson, Canon, or Hewlett-Packard, its icon doesn't show up in the Chooser until you run the installer program on the CD that accompanies the printer. For example, if you install the software for an Epson Stylus 740 inkjet printer, the Chooser now displays an icon called SC740 (which is the Epson programmers' cryptic way of saying "Stylus Color 740").



Select a PostScript Printer:

Silentwriter 95
MP6/Hallway
Design Dept. Color Laser
Hi-Res Turbo 600 dpi

AppleTalk [✓ Active] Inactive

Figure 19-1:
The Chooser lets you specify which printer you want to use. At left: The various drivers (software translators) for some printers the Mac knows about. At right: the list of available laser printers–or, if you have a non-network printer, a list of ports to which that printer might be connected.

Tip: Each icon in the Chooser is represented by a piece of software in your System Folder→Extensions folder. Feel free to discard the icons of printers you don't imagine you'll be using. If you think it's unlikely that you'll be printing on, say, an ImageWriter printer (which was discontinued in 1996), by all means throw away its icon. (Keep the LaserWriter 8 icon on hand, however; it's the driver software for almost any brand of laser printer.)

Before making any printouts at all, in other words, you must first install the software for your new printer—and then you must open the Chooser and click the corresponding icon. On the right side of the screen, you may have to make another click:

- If you click the icon for a networkable office laser printer (such as the LaserWriter 8 icon), and you are, in fact, connected to a network, the names of all operational office printers appear on the right side of the window. (If your corporate network is so big that it has multiple *zones,* first click the appropriate zone at the lower-left of the Chooser window.) Click the name of the printer you'd like to use.
- If you click the icon for a non-networkable printer, such as an inkjet printer from Epson, Canon, or Hewlett-Packard, the right side of the screen offers a list of Mac connectors to which that printer might be attached. If you have a modern Mac, the inkjet printer is probably connected to your USB socket, and you have no further choices to make. If you have an older Mac, you must specify whether the printer is connected to the modem or printer port.

Note: If you intend to connect your inkjet printer to the printer port, the controls at the bottom of the window should say *AppleTalk Inactive*. AppleTalk should be active *only* when you want to connect a networkable laser printer to your printer port.

- If you click the LaserWriter 8 icon, you can, if you like, also click the Setup button on the right side of the screen. Now you're offered an enormous list of laser printers (exclusively discontinued Apple models, unless you've bought a new laser printer and run its installer). By double-clicking the one you actually own, you can inform your Mac of that printer's special capabilities—the largest paper it can handle, double-sided printing features, additional paper trays, and so on. (If your printer doesn't show up in this list, click Generic.)

In any case, your primary mission, upon installing a new printer, is to select its name in the Chooser and then close the window. (You can also use the Desktop Printing Utility program, described later in this chapter, to set up a new printer.) Now you're ready to make printouts, as described in "Making the Printout," also later in this chapter.

FREQUENTLY ASKED QUESTION

The Forgetful Chooser

Every time I open the Chooser, I have to click my printer icon again. Why can't it remember what I chose the last time?

Actually, it can, and it does. If you try printing another document, you'll discover that the Mac uses the same printer again—proof that the Chooser does indeed remember your choice.

What's probably confusing you is the fact that when you open the Chooser again later, the printer *icon* is no longer selected. That's standard behavior. As it turns out, the Chooser is the command center for more than just printers. Sometimes you use it to configure your fax modem; other times you use it to connect to your network, as described in Chapter 16.

Each time you open it, in other words, the Mac doesn't know what settings you intend to change—printer, fax modem, or network—so it highlights *nothing*. Behind the scenes, however, the Mac is still perfectly aware of your preferred printer.

(On the other hand, if, each time you print, you get a *message* that instructs you to choose a printer in the Chooser, then the Mac really *is* forgetting your printer choice. It's likely that your Mac's built-in 5-year lithium battery, whose purpose is to maintain such settings when the computer is turned off, has died. An Apple dealer can replace it for you for about $25.)

Desktop Printer Icons

If you work alone, with one printer attached to one Mac, you can afford to skip this section.

But some people have more than one printer. You might have a laser printer attached to your Ethernet port, for example, and a color inkjet attached to your USB port. You may also have a choice of printers if you work in a networked office—maybe an old black-and-white Apple LaserWriter, plus an expensive color laser printer for printing mockups of brochure designs. Either way, you need a way to specify, on a printout-to-printout basis, which printer you want to use.

Of course, you can always open the ‍→Chooser whenever you want to switch printers, and then click the corresponding icon. If you switch infrequently, this method works perfectly well.

Note: If your printer isn't an Apple model, using the Chooser may be the *only* way you can switch printers; not every printer comes with software that's compatible with the Desktop Printing feature described below.

If you switch more often, however, you may wish for a more convenient method of switching printers. That's why Apple created its Desktop Printing feature: as shown in Figure 19-2, you can set up your desktop with an icon representing each printer. You also wind up with a Printing menu that unlocks several additional printing options.

Even Mac users with only a single printer sometimes encounter this feature. They wind up with a big bold desktop icon representing that one printer, and have no idea what it's for or how it got there. Now you know.

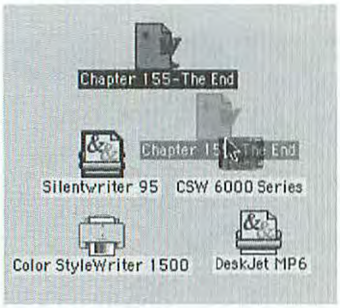


Figure 19-2:
You can direct each printout to a particular printer just by dragging its icon onto the corresponding desktop printer icon. When you print using the File→Print command, the printer with the big bold icon (top left) is the one that does the job.

FREQUENTLY ASKED QUESTION

The Xed-Out Desktop Printer Icon

I've got a big bold printer icon on my desktop, like the ones you describe. But it has a huge X through it. What's it for?

The Desktop Printing software described in this section requires two extensions in your System Folder→Extensions folder, called Desktop Printing Extension and Desktop Printer Spooler. If you turn these extensions off, you cut out the system software's communication with the printer icons on your desktop—the Mac responds by displaying the X. It means, "This desktop printer icon isn't working at the moment, but when you turn your extensions back on, all will be well."

You'll encounter this syndrome not just when you move or turn off the desktop printing extensions, but also whenever you press the Shift key while the Mac is starting up. Doing so turns off all extensions, *including* the two Desktop Printing ones.

OK, you're probably going to say that this is a related question—but how can I get rid of those desktop icons entirely?

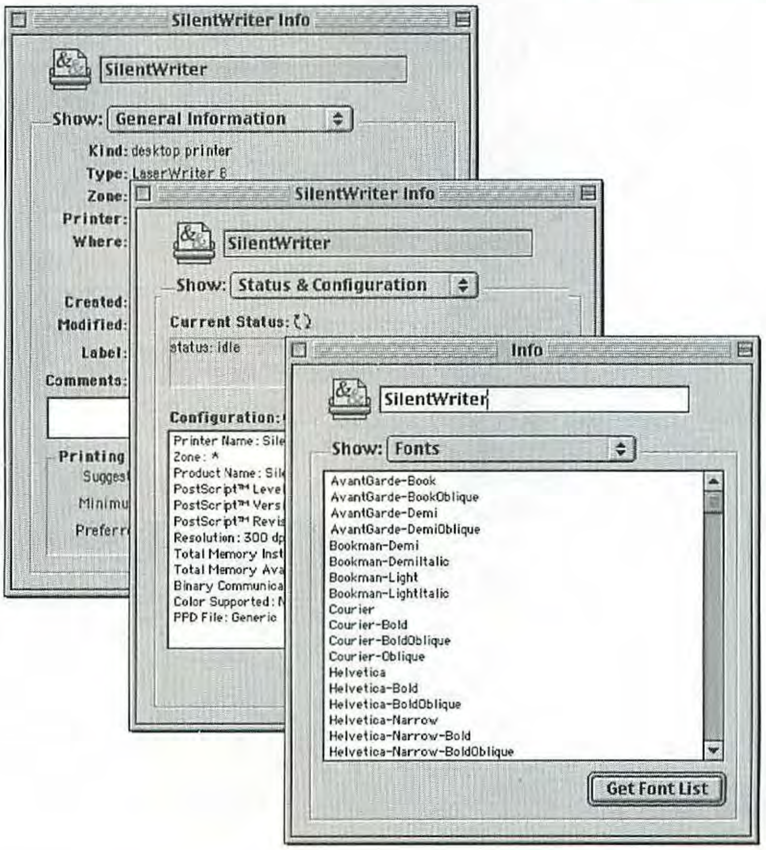
This is a related question. As you may have discovered, you can't ditch desktop printer icons by dragging them to the Trash; the Mac recreates at least one of them automatically. You must first turn off the two desktop printing extensions (using Extensions Manager, described in Chapter 12) and then restart the computer. Only then can you throw out your desktop icons, which now appear with Xes on them.

Creating Desktop Printer Icons

If you found a printer icon already on your desktop after installing Mac OS 9, it's probably because you patiently answered questions posed by the Mac OS Setup Assistant that runs just after installation of Mac OS 9. (One of those questions asks what kind of printer is connected to your Mac.) You may also wind up with an automatic desktop printer icon after upgrading to Mac OS 9 from an earlier OS version, in which you had already selected a printer.

Creating another desktop printer icon is as easy as choosing →Chooser and clicking another printer icon. Each time you do so and then close the Chooser, another printer icon appears on your desktop. (The exceptions, as noted earlier, are the icons for non-Apple printers that still aren't compatible with the desktop-printer feature.)

Once you've got a few printer icons, they don't necessarily have to remain *desktop* printer icons. You can move these icons anywhere you like—for example, into a folder, into the menu, or someplace else where they create less clutter. You can rename them, too—click the name once, and then type the replacement name, exactly as you would rename any Mac icon.


```
SilentWriter Info
SilentWriter
Show: General Information
Kind: desktop printer
Type: LaserWriter 8
Zone:
Printer:
Where:
Created:
Modified:
Label:
Comments:
Printing
Suggest
Minimu
Preferr

SilentWriter Info
SilentWriter
Show: Status & Configuration
Current Status:
status: idle
Configuration:
Printer Name: Sile
Zone: *
Product Name: Sil
PostScript™ Level
PostScript™ Vers
PostScript™ Revis
Resolution: 300 dp
Total Memory Inst
Total Memory Ava
Binary Communica
Color Supported: N
PPD File: Generic

Info
SilentWriter
Show: Fonts
AvantGarde-Book
AvantGarde-BookOblique
AvantGarde-Demi
AvantGarde-DemiOblique
Bookman-Demi
Bookman-DemiItalic
Bookman-Light
Bookman-LightItalic
Courier
Courier-Bold
Courier-BoldOblique
Courier-Oblique
Helvetica
Helvetica-Bold
Helvetica-BoldOblique
Helvetica-Narrow
Helvetica-Narrow-Bold
Helvetica-Narrow-BoldOblique
Get Font List
```

Figure 19-3: *When you choose File→Get Info, you get a screen of useful information about the printer whose icon you highlighted (left). If you're investigating a laser printer, use the Show:→Status & Configuration pop-up menu command (middle), which gives the statistics about the printer, such as whether it's color or black-and-white, its resolution (sharpness, as measured in dots per inch [dpi]), and so on. Or you can choose Fonts, which—after you click Get Font List—shows you a list of typefaces that are pre-installed on the printer (bottom).*

If you ever forget a printer's original name, highlight its icon and then choose File→Get Info. As shown in Figure 19-3, printer icons have Get Info windows all their own; the first one, General Information, identifies the printer's original information.

Using Desktop Printer Icons

Despite their innocent appearance, desktop printer icons are teeming with sophisticated features for managing your printouts.

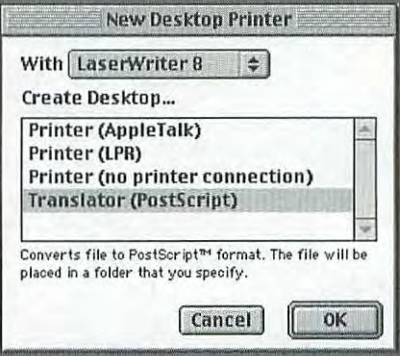
Choosing your favorite

If multiple printers are indeed connected to your Mac, the first thing Desktop Printing lets you do is indicate which you'd like to use for your next printout. To do so, click once on its icon. When you do so, a new menu appears on your menu bar called Printing. The commands it offers depend on what kind of printer icon you've clicked—for example, if you clicked an inkjet printer, the Printing menu lets you specify how the printer is connected to your Mac; if you clicked a laser printer, you get a Setup command that duplicates the functions of the Setup button in the Chooser; and so on.

POWER USERS' CLINIC

Desktop Printer Utility

In your Apple Extras→Utilities→Apple LaserWriter Software folder is a forgotten little program called Desktop Printer Utility. As shown here, it offers a method of creating special mutant breeds of desktop printer icons.



Converts file to PostScript™ format. The file will be placed in a folder that you specify.

The first kind, Printer (AppleTalk), creates a standard laser-printer desktop printer icon, exactly as though you had clicked LaserWriter 8 in the Chooser. The next item, Printer (LPR), is for office networks that rely on TCP/IP (the protocol used by the Internet); it creates a printer icon for a printer elsewhere on your TCP/IP network. (If you're a network administrator who set this up, you'll know what this is about.)

A Printer (no printer connection) icon doesn't print anything; it's just a holding tank, like a desktop printer that's been put on permanent hold. When you're using your laptop at 39,000 feet, you can freely use your Print command. The laptop will do the dirty work of converting your documents into printout files, which this icon houses. Later, when you return to the ground—or someplace where there's an actual printer—you can drag the pent-up printout icons from this "printer's" window onto one of your other desktop printer icons to get the hard copy.

Finally, the Translator (PostScript) desktop printer type is a converter. When you use this kind of "printer," the Mac converts your printed file into a PostScript file, as described later in this chapter.

After selecting the kind of special printer icon you want, you face another dialog box in which you must specify additional settings—for the TCP/IP printer, for example, you must enter the printer's network address. Finally, click Create, and then save the new icon with the location—and the name—you want it to have.

ROKU EXH. 1002

But regardless of the kind of printer icon you clicked, the Printing command always contains a Set Default Printer command. That command, or its ⌘-L keyboard shortcut, establishes the highlighted printer as the *default printer*—the Chosen One. A thick black border appears around its desktop icon, letting you know that your next printout will go to that printer.

Actually, though, there are three faster ways to choose a new favorite printer:

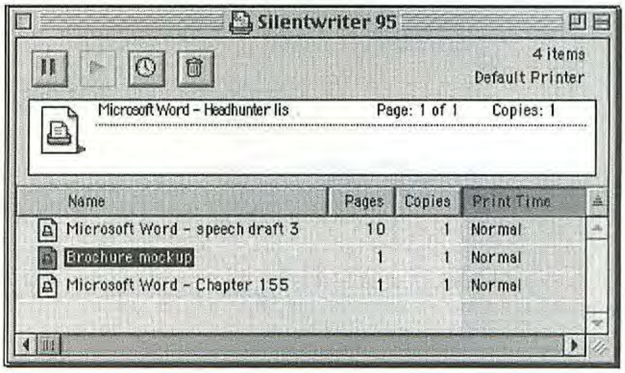
- Control-click the desktop printer icon; choose Set as Default from the pop-up menu.
- If your Control Strip is open, as described in Chapter 4, choose a printer's name from the Desktop Printing tile. A dot appears in the pop-up menu beside the name of the currently selected printer.
- Just drag a document icon (one that you want to print) onto a desktop printer icon, which automatically becomes the new default printer.

Managing printouts

The real fun of desktop printer icons only begins when you've actually tried printing something. (To print something, see "Making the Printout," later in this chapter; it boils down to opening a document and then choosing File→Print.)

Once you've done so, an amazing thing happens: your chosen desktop printer icon suddenly behaves like a folder that contains the printouts-in-waiting. Double-click one of these icons to see something like Figure 19-4: the printouts that will soon be sliding out of your printer appear in a tidy list.

Figure 19-4:
Waiting printouts show up in a desktop-printer window. You can manipulate this window exactly as you would any Finder list view: for example, you can sort the list by clicking the column headings Name, Pages, and so on; make the columns wider or narrower by dragging the column-heading dividers horizontally; or reverse the sorting order by clicking the triangle button above the scroll bar.



Name	Pages	Copies	Print Time
Microsoft Word - speech draft 3	10	1	Normal
Brochure mockup	1	1	Normal
Microsoft Word - Chapter 155	1	1	Normal

There's no end to the control you now have over these waiting printouts, which Apple collectively calls the *print queue:*

- **Rearrange them.** By dragging the names of your printouts up and down in the list, you can specify which ones get printed first—namely, the ones at the top of the list. (This works only when you're viewing the list sorted in Print Time order, as shown in Figure 19-4.)

- **Delete them.** By clicking an icon, or Shift-clicking several, and then clicking the tiny trash icon at the top of the window, you remove items from the list of waiting printouts. If you prefer, you can also drag icons directly into the real Trash in the corner of your screen. There they become separate printout icons, which you can truly delete—by emptying the Trash—or rescue by dragging back onto a desktop printer icon.
- **Duplicate them.** After highlighting the name of a waiting printout, you can choose File→Duplicate to make a copy of it in the same window. Now you'll get two printouts of the same thing.
- **Add to them.** You can drag document icons—Word or AppleWorks documents, for example—directly onto a desktop printer icon (or into its open window). Doing so prints the document and establishes that printer icon as the new default printer, as noted above.
- **Change your mind.** By highlighting a printout and then clicking the Pause button (the leftmost of the four VCR-style icons above the list), you put that printout on hold. It won't print out until you highlight it again and click the Play button (the second icon at the top of the window).
- **Schedule them.** Click one of the printouts and then click the tiny clock icon above the list. As shown in Figure 19-5, you can reschedule this printout for a time when the printer isn't so busy, as a favor to those co-workers with more urgent print jobs.

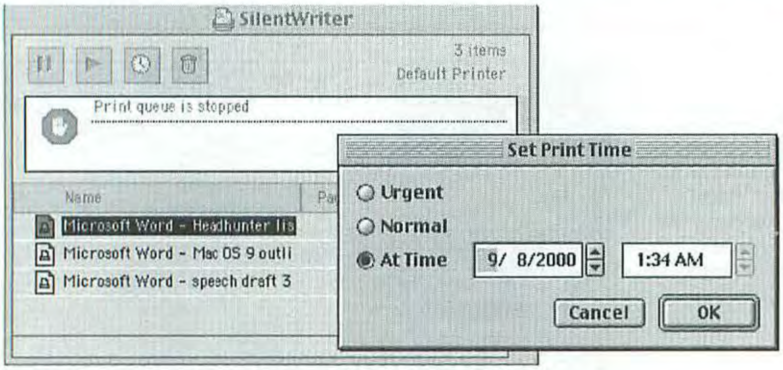


Figure 19-5: *Choose Urgent to force your printout to the top of the list, so that it will be printed first. Or click At Time to schedule your printout for some later time—so that your 75-page thesis prints out in the middle of the night to avoid tying up the printer, for example.*

- **Halt them all.** You can stop all printouts for a specific printer (whose window is open) by choosing Printing→Stop Print Queue. (Choose Printing→Start Print Queue to re-enable printing on this printer.)

Tip: If menus aren't your thing, hold down Shift and Option to make the Pause button turn into a tiny stop-sign icon button, which you can click to stop the print queue. Hold down the same two keys to make the Play button take on the shape of a printer icon, which means Start Print Queue.

ROKU EXH. 1002

- **Switch them to another printer.** If this is the day your 75 co-workers are all printing out their résumés, you can drag your printout-in-waiting onto the icon (or into the window) of another desktop printer, thus forcing your work to print out on a different printer. (This works only if you're dragging it to the icon of the same *kind* of printer—from one laser printer icon to another, for example.)

Printing Without Desktop Icons

Suppose you've got one Mac and one printer, and you'd really rather not get into the complexity of the desktop-printer icons described in the previous section. And yet, having scanned the preceding pages, you're feeling left out—why shouldn't you, too, be able to rearrange printouts, put printouts on hold, and perform other kinds of printout manipulation?

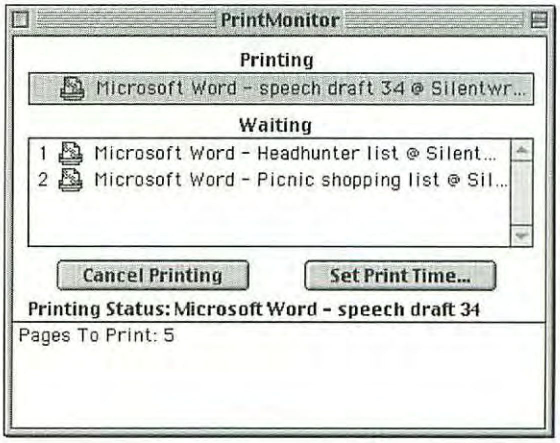
Actually, you can. If you regularly use *background printing,* described in the next section, you get most of the same benefits as the desktop-printing feature, even though you use only a single printer.

Tip: You won't encounter PrintMonitor, the program described in the following section, or any of its features when the desktop printing feature described in the previous section is turned *on.* To turn Desktop Printing off, choose →Control Panels→Extensions Manager, turn off the icons called Desktop Printing Extension and Desktop Printer Spooler, and then restart the Mac.

Meet PrintMonitor

When you print a document, the Mac does two things. First, it launches a printing-management program called PrintMonitor. Many people aren't even aware that this program has been launched, but if you check your Application menu about 10 seconds after printing something, you'll see PrintMonitor in the list of open programs.

Figure 19-6:
You can rearrange the sequence of printouts waiting to happen by dragging their names up or down in the Waiting area of the PrintMonitor window. If you click Set Print Time, you can put a printout on hold or reschedule it for a less busy printing time.



Microsoft Word - speech draft 34 @ Silentwr...

1 Microsoft Word - Headhunter list @ Silent...
2 Microsoft Word - Picnic shopping list @ Sil...

Printing Status: Microsoft Word - speech draft 34

Pages To Print: 5

Second, the Mac creates a *spool file*—a printout-in-waiting, exactly like the ones described in the previous discussion. Most of the time, however, you never see these spool files. They're buried in the System Folder→PrintMonitor Documents folder—and even then, only for a moment. Once the Mac prints these documents, they disappear from the PrintMonitor Documents folder forever.

Controlling PrintMonitor Printouts

To gain some control over the printout process, choose PrintMonitor from your Application menu. As shown in Figure 19-6, this funny little program pops to the foreground, showing a list of the documents waiting to be printed.

Before you know it, your Mac will actually print these documents, and you'll have lost whatever control you hoped to gain. But if you act quickly, you can manipulate the printouts like this:

- **Rearrange them.** Drag printouts up or down the list to make them print sooner or later than other documents.
- **Cancel them.** Click a waiting printout's name and then click Remove From List if you change your mind about printing it.
- **Reschedule them.** Click a waiting printout and then click Set Print Time. You'll be offered the chance to specify a later date for printing this document. You can also click Postpone Indefinitely, which puts the printout on hold until you highlight the printout again, click Set Print Time again, and indicate a printing time.
- **Collect printouts for later.** Even when there's nothing listed in the PrintMonitor window, you may enjoy this trick: While the PrintMonitor window is open, choose File→Stop Printing. Now, whenever you use the Print command, the Mac will convert your document into a printout-to-be, but won't actually attempt to send it to a printer.

  As described in "Using Desktop Printer Icons," earlier in this chapter, that can be a great feature when you're on the plane with your laptop, for example; when you touch down and connect to the printer, you can choose File→Resume Printing. All of your saved printouts will tumble out in short order.

Tip: PrintMonitor normally appears only when you're actually printing something. You might wonder, therefore, how you can get to its menus—in order to choose Stop Printing, for example—without having to make a printout of something.

Fortunately, PrintMonitor is a genuine program with a genuine double-clickable icon. To find it, open your System Folder→Extensions folder. You can double-click the PrintMonitor icon just as you would any application.

PrintMonitor Preferences

While PrintMonitor is open, you can also choose File→Preferences. A dialog box appears, offering these choices:

- **Show the PrintMonitor window.** Actually, the No/Yes options here have no effect; PrintMonitor *always* launches in the background when you make the printout, regardless of the setting you make here.
- **When a printing error needs to be reported.** If the printer is out of paper or ink, for example, how urgently do you want PrintMonitor to try to get your attention? You can have it display a message, make your Application menu blink, or just display a tiny black diamond beside the PrintMonitor name in the Application menu. (You won't even see that diamond until you actually open the Application menu.)
- **When a manual feed job starts.** Suppose that your printer has a "manual feed" option (which takes one sheet at a time from a special tray or slot), and that you've selected that option when making your printout. How do you want PrintMonitor to say, "It's time to put in the next sheet"? Once again, you can specify how much flashing or message-showing PrintMonitor should show you.

Making the Printout

This chapter so far has covered the elaborate printout-manipulation features built into Mac OS 9. You're not compelled to use any of these fancy options; many Mac users don't. Almost everyone, however, sooner or later makes a basic printout. You can print your documents either from within the programs you used to create them—or directly from the Finder.

Printing from Within Your Programs

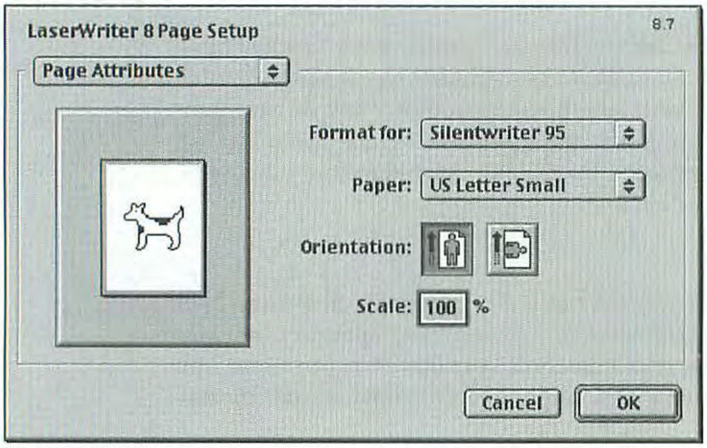
The experience of printing depends on the printer you're using—laser printer, color inkjet, or whatever. In every case, however, all the printing options hide in two commands: Page Setup, which you need to adjust only occasionally, and Print. You'll find these two commands in the File menus of almost every Macintosh program in existence—Word, Excel, AppleWorks, Photoshop, your email program, your Web browser, and on and on.

Page Setup

Most people use this command only when they want to print a document rotated sideways on the page, so that it prints "the long way." But even some of the less-used options here are sometimes useful (see Figure 19-7). The Scale or Scaling control, for example, lets you reduce or enlarge your document, which can be handy if the program you're using doesn't offer such a control.

The remaining choices vary; the Page Setup options for an Epson inkjet, for example, differ dramatically from those for a laser printer. Only your printer's user manual can tell you exactly what these choices do. Because most laser printers offer the standard LaserWriter 8 options, here's a rundown of some of the most useful. (They're listed here by the corresponding command in the pop-up menu at the upper-left corner of the dialog box.)

- **Page Attributes.** Use the Paper pop-up menu to specify what size paper you're printing on—US Letter, US Legal, or one of the standard European paper sizes (A4 and B5). Don't be confused by the "small" variants listed here (such as US Letter Small); these paper *dimensions* are identical to the non-small versions. The only difference is the margin: if you turn on the Small option, the laser printer chops off any part of the printout closer than half an inch to the edge of the page. (The non-small page sizes can get to within a quarter of an inch.)



Format for: Silentwriter 95
Paper: US Letter Small
Orientation:
Scale: 100 %

Figure 19-7: *The options included in this dialog box depend on the printer model you're using. Sometimes, as when printing on a laser printer, the pop-up menu at upper-left lets you switch among different screens full of choices. The effect of each printing option is illustrated by the little animal in the dialog box, which is known as the Dogcow. (The rumor that the Dogcow is related to the animal on the cover of this book is pure speculation.)*

Tip: Clearly, "US Letter Small," with its fatter non-printable margin, is usually less useful than "US Letter." Who wants to risk having printouts chopped off at the margins? Yet your Mac insists on using the Small option, no matter how many times you change this pop-up menu.

There is a secret, however, to changing the setting once and for all: Option-click the OK button. You'll be asked if you're sure you want your new Page Setup options preserved; click Save.

(Note: You must perform this ritual once for each program that offers its own Page Setup options, as described in the next paragraph–and again, once, for all *other* programs.)

- **Microsoft Word.** To see a window full of printing options unique to the program you're actually using—Word is just one example—choose its name from the pop-up menu. (If you don't see your program's name listed in this pop-up menu, then this program offers no special options of its own.)

- **PostScript Options.** The vast majority of these checkboxes are novelty items or leftover workarounds from the days when most Mac users printed from MacWrite and MacPaint. Most of the time, you should turn them all off.

  For example, **Flip Horizontal** prints out a mirror image of your document—useful if you intend to create iron-on logos, read the printout in a mirror or reflected off of your car windshield, and so on. **Invert Image** creates a black-on-white, negative image, suitable for Halloween posters and using up your printer cartridge quickly.

- **Unlimited Downloadable Fonts** can be useful when you're printing on a laser printer with limited memory—especially if it's a document containing lots of different fonts. Then, instead of overwhelming the printer's memory by trying to transmit too many fonts, the Mac instructs the printer to "forget" one set of fonts before receiving the next. All of this makes the printout take much longer to appear; furthermore, this process can wreak havoc (or, more precisely, wreak Courier) with the fonts embedded in EPS graphics.

The Print command

Although you can grow to a ripe old age without ever seeing the Page Setup dialog box, you can't miss the Print dialog box. It appears, whether you like it or not, whenever you choose File→Print in one of your programs.

Once again, the exact options you encounter depend on the printer you're using (or more specifically, they depend on the driver you've selected in the Chooser). You're always offered a few standard options, however, including these:

- **Copies.** Type the number of copies you want printed.
- **Pages.** You don't have to print an entire document—you can print, say, only pages 2 through 15.

Tip: You don't have to type numbers into both the "From:" and "To:" boxes. If you leave the first box blank, the Mac assumes that you mean "from Page 1." If you leave the second box blank, the Mac understands you to mean "to the end." To print only the first three pages, in other words, leave the first box blank, and type 3 into the second box. (These page numbers refer to the physical pages you're printing, not to any fancy numbering you've set up in your word processor. As far as the Print dialog box is concerned, the first printed page is Page 1, even if you've told your word processor to label it page 455.)

If you have a color inkjet printer, you can also specify what print quality you want, what kind of paper you're printing on, and so on.

If you're using a laser printer, you get dozens of additional options. They're divided into separate screens, accessible using the unnamed pop-up menu just below the "Printer:" pop-up menu. Many of these options are extremely technical, designed for use by graphic designers and print shops; fortunately, you can choose Help→Show Balloons and then, by pointing to each option, read a description of its function.

In the meantime, a few of the options can occasionally be useful. For example:

- **General.** Specify the number of copies, range of pages you want printed, and whether or not you intend to print one page at a time (Manual Feed).
- **Microsoft Word.** Whatever program you're using—Word, AppleWorks, or anything else—may offer its own special printing options on this screen.

- **Background Printing.** You can read about background printing in the upcoming sidebar called "Background Printing Basics." The beauty of using a laser printer, however, is that you can turn background printing on or off independently for *each* printout, using these controls. (You can also specify when you want this document printed.)

- **Color Matching.** Use these controls to specify how you want the colors in your document, if any, translated on their way to the printer. For example, you can print a color document in black-and-white even on a color printer, if that's what you want. (Unfortunately, Apple's programmers have yet to devise a way to make a color printout on a *black-and-white* printer.)

  If you choose ColorSync Color Matching (described later in this chapter) or PostScript Color Matching, you can also use the Intent pop-up menu. These options are designed to address a characteristic problem of color printers: they can't actually reproduce every color in the rainbow. That could be a problem if you're printing a photograph of, for example, a rainbow.

  You can choose from among the various visual/psychological compromises here. For photos, Perceptual Matching is usually best; for graphs and charts, Saturation Matching may be more successful. If you use the "Auto selection" option, the Mac automatically uses Saturation matching for graph-like drawings (object-oriented artwork), but Perceptual for photos (bitmapped artwork).

  When you choose one of the color matching choices from the Print Color pop-up menu, you can also specify a printer profile; this, too, is described later in this chapter under "ColorSync."

- **Cover Page.** If you turn on this option (by clicking **Before Document** or **After Document**), the Mac will tack an extra page onto your printout. It bears the name of your file, your name, and the time. The **Paper Source** pop-up menu lets you specify which of your printer's paper trays you want to use for the cover page—so that you can stock that tray with scrap paper, for example.

- **Font Settings.** If you plan to save your file as a PostScript file, as described in "Save as File," below, **Annotate Font Keys** embeds textual comments to your PostScript file that describe the fonts you used—which can be useful in times of troubleshooting. The other settings here affect what happens when you use a typeface in your document that isn't one of the fonts built into your printer. For example, the **Preferred Format** option applies only when your Mac has both TrueType and PostScript versions of a particular font installed.

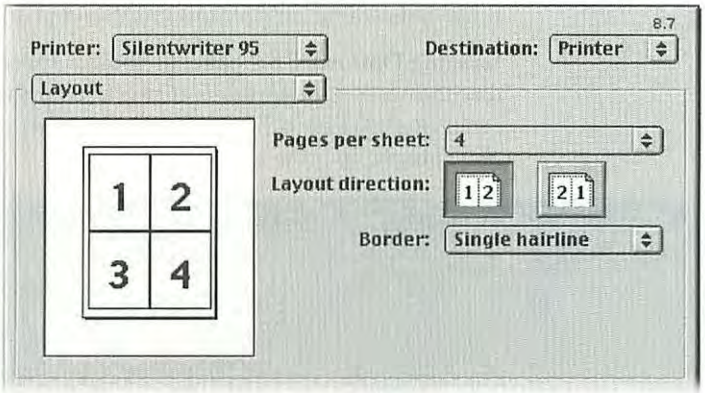
- **Job Logging.** Using these controls, you can specify what you want to happen when the laser printer reports a PostScript error (when your document is too complex for the printer's memory, for example)—whether you want a short message to appear on the screen, or a more detailed and technical message to print out.

  You can also ask your Mac to keep a record of the printouts it makes. (Specify where you want the file kept by clicking the Change button.) You can choose from

Generate Job Copy, which keeps a duplicate of the actual printout in the journal, or Generate Job Log, which just makes a journal of what was printed when.

- **Layout.** As shown in Figure 19-8, these options can be very useful. For example, you can save paper and toner cartridges by printing several miniature "pages" on a single sheet of paper.

Figure 19-8:
By asking the Mac to print several pages per sheet of paper, you can compare various designs, look over an overall newsletter layout, and so on. Using the Border pop-up menu, you can also request a fine border around each miniature page. Some printers even offer a Print on Both Sides option here, so that you can print little booklets.



- **Save as File.** Using the Destination→File pop-up menu command at the top of this dialog box, you can turn your printout into a file on your hard drive rather than sending it to the printer. You can then send the resulting file—a *PostScript file*—directly to a print shop. This file contains a complete, self-contained computer-language description of your printout that the print shop can use to print your document without requiring the program you used to create it—or even the document itself.

  The options on the Save as File screen let you describe the file you'll be creating. Use the **Format** pop-up menu, for example, to specify whether you want a standard PostScript file, as described above, or an *EPS* (Encapsulated PostScript) graphic that you can insert into a page-layout program. (In general, choose EPS Mac Enhanced Preview, which is in color, from the pop-up menu here; the Standard Preview is in black and white, and the No Preview doesn't let you see the image at all when placing it into your page-layout program.)

Tip: If you have the software called Acrobat Distiller, the Format pop-up menu offers another choice: Acrobat PDF. This choice lets you create an Adobe Acrobat document–a file that any Mac or Windows user can view, read, and print using the free Acrobat Reader program included with every Mac and PC. (You must buy Adobe Acrobat 4 to gain this option–or get it with the purchase of a program like Adobe PageMaker or InDesign.)

The **PostScript Level** command specifies what kind of laser printers you intend your file to be printed on (keeping in mind that Level 1 printers—very old ones—offer fewer advanced graphics features, such as color handling and patterns).

Use the **Font Inclusion** pop-up menu to answer the question: "What happens if I used fonts in my document that aren't built into the printer that will be printing this PostScript file?" If you choose None, you'll get the Courier typeface in place of your special fonts. If you click All, you won't have that problem—but the resulting PostScript file will be gargantuan, taking up lots of disk space; it will incorporate every character of every font in your document. Choosing All But Standard 13 instead helps the problem a little bit, by omitting from your file the standard 13 fonts that are built into almost every laser printer (Times, Helvetica, and so on).

- **Imaging Options.** This panel of choices appears only if you're using one of the discontinued Apple printers that offered printout-enhancement features called FinePrint and PhotoGrade. Experiment with these options to see what gives your text and graphics the best printouts.

UP TO SPEED

Background Printing Basics

In the beginning, there was no such thing as background printing. The Mac simply locked you out whenever it was printing. You couldn't do anything with the Mac except stare at the "now printing" message.

With the invention of background printing, your options are more interesting. Background printing lets you keep using your Mac while the printing takes place. It hands the printing tasks off to a special program—PrintMonitor or Desktop PrintMonitor—that feeds the printout, a little bit at a time, to your printer, in the tiny pauses between your keystrokes and mouse clicks. As a result, the printout takes longer—but you can keep working in the meantime.

You can turn background printing on or off fairly easily—once you find the control, whose location depends on the printer in question. If you're using a laser printer, for example, you turn background printing on or off independently for each printout. Just choose Background Printing in the dialog box that appears when you choose File→Print, as described in this chapter. (Unless you intervene, background printing is always on.) Similarly, if you're using an Epson inkjet, click the fourth of the five icons in the Print dialog box—the one whose icon is a clock superimposed on a printout—to turn background printing on or off.

For most Apple non-laser printers, you can't turn background printing on or off one printout at a time—only for the entire printer. To do so, choose →Chooser, click the icon for the printer, and then click the Background Printing On or Off button that appears on the right side of the Chooser window.

Printing from the Finder

You don't necessarily have to print a document while it's open in front of you. You can, if you wish, print it directly from the Finder, using one of these three methods:

- Highlight the document icon, and then choose File→Print.
- Control-click the document icon; choose Print from the contextual menu that appears.
- If you're using the desktop-printer icons described at the beginning of this chapter, drag the document icon onto one of these printer icons.

Now the Mac launches the program that created it—Word or AppleWorks, for example—and the Print dialog box appears, so that you can specify how many copies you want and how many of the pages you want printed. When you click Print, your printer springs into action, and then the program quits automatically (if it hadn't already been open).

Tip: If you're using a laser printer, and you print a PostScript, EPS, JPEG, or PICT graphics document, the Mac doesn't have to launch the parent program. Instead, the Mac sends the file directly to the printer, saving you a little bit of time and itself a little bit of memory.

You can also highlight several icons and print them simultaneously, even if you used different programs to create them. (If they're in different windows, consider dragging their icons onto the desktop, so that you can highlight them all simultaneously. Then, after the printing is over, choose File→Put Away, which makes the icons jump back into their original folders.)

Tip: The Finder's File menu also contains commands for printing Finder windows–Page Setup and Print Window (or, if no window is open, Print Desktop). You may find the Print Window command useful when you want a quick table of contents for, for example, a Zip disk. It's also very useful when you're trying to find out whether or not a printer is working–just open any window and then choose Print Window. You save the time and effort of finding some test document to print.

Fonts

Over the years, Macintosh fonts have improved considerably. No longer do you have to pray to the printer gods that your beautiful flyer won't come out with jagged-looking typeface because you chose the wrong font *type,* one that doesn't have smooth edges. (Jagged-printing *bitmapped fonts* were standard on original Macs, but have now, mercifully, almost completely disappeared from the Earth.)

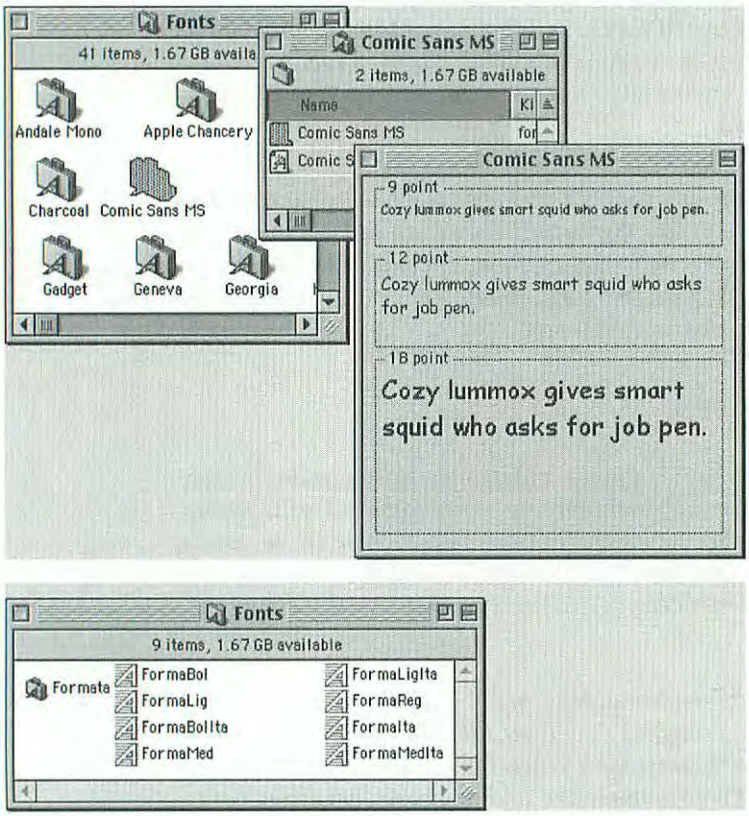
TrueType Fonts

If you stick to the fonts that came with your Mac (and the additional ones installed by AppleWorks, Microsoft programs, and so on), your font life is fairly uneventful. Whether you know it or not, every font in your System Folder→Fonts folder is what's known as a *TrueType* font. TrueType fonts are easy to use and easy to understand; no matter what point size you select for these fonts, they look smooth and professional, both on the screen and when you print. Figure 19-9 reveals the simplicity and common sense in the design of TrueType fonts.

PostScript Fonts

If you're a graphic designer, however, you may well have inherited or purchased fonts that come in another format: *PostScript* fonts. These fonts have several advantages: first, in the professional publishing world, they're everywhere; the PostScript fonts listed in the Adobe catalog, for example, number in the thousands. Second, if you send your finished work to a print shop, PostScript fonts will be met with a friendlier reception, because TrueType fonts can choke some older typesetting equipment.

But PostScript fonts have some considerable disadvantages, too. For example, each comes in two pieces—one that provides the information necessary to display the typeface on the screen, and a set of additional files required by your printer. (Figure 19-9 illustrates this clutter in your Fonts folder.) If you print a document for which some of these printer files are missing, you'll get the wrong font in your printout, even if the document looks fine on the screen.



Cozy lummox gives smart squid who asks for job pen.

Figure 19-9: *All of your fonts sit in the System Folder→Fonts folder. Each TrueType font is self-contained in a single font "suitcase" (like those shown in the top illustration). You can double-click a TrueType font's suitcase to see which custom-tailored point sizes are available, if any (top middle), and then double-click one of the icons inside to see what the font looks like (top right). Each PostScript font (like the one font shown at bottom), meanwhile, requires several different files. You need a font "suitcase" (to contain the screen fonts), which you can double-click to view the font, just as you can with a TrueType font. You also need a set of cryptically abbreviated printer fonts.*

But your document probably *won't* look fine on the screen unless your Mac has Adobe Type Manager (ATM), a special control panel. Without it, PostScript fonts that you use in your documents at nonstandard sizes (anything other than, for example, 9, 10, 12, 14, 18, or 24 points) appear jagged on the screen.

Fortunately, ATM is easy to get—and it's free: just install the free Acrobat Reader program. Acrobat Reader is on your Mac OS 9 CD, or you can download the latest version from *www.adobe.com.* (Acrobat Reader is designed to let you read the PDF, or Adobe Acrobat, files, which are becoming increasingly popular on the Internet—especially as user manuals for downloaded software.)

Managing Your Fonts

As shown in Figure 19-9, every font that appears in the Font menus of your various programs is represented in your System Folder→Fonts folder by an icon—or several.

Installing fonts

Mac OS 9 comes with 18 great-looking TrueType fonts: Apple Chancery, Capitals, Charcoal, Chicago, Courier, Gadget, Geneva, Helvetica, Hoefler Text, Monaco, New York, Palatino, Sand, Skia, Symbol, Techno, Textile, and Times. (When you install Microsoft Internet Explorer, you inherit another handful, including Andale, Arial, Impact, Trebuchet, Verdana, and the symbol fonts Webdings and Wingdings.) But the world is filled with additional fonts. You may find them on the CD-ROMs that come with Mac magazines, on Mac software Web sites, or in the catalogs of commercial typeface companies.

To install a new font, drag its suitcase (if it's TrueType) or its suitcase *and* accompanying printer files (if it's PostScript) either into the Fonts folder or directly onto the System Folder icon. (In the latter case, the Mac offers to install them into the Fonts folder automatically. You'll be notified that the newly installed fonts won't appear in your Font menus until the *next* time you launch your various programs.)

Removing fonts

When you've had enough of a font, you can remove it from your Mac. To do so, quit all running programs. Then open the System Folder, open the Fonts folder, and drag the offending font suitcase into any other folder (or the Trash). Now when you launch your programs, that font will be absent from the Font menus.

More about font suitcases

It's Mac OS 9's least publicized new feature: your Fonts folder can contain up to 512 font suitcases—quadruple the limit in previous OS versions—each of which can contain either TrueType fonts or the screen half of PostScript fonts.

If you, owner of more fonts than Felix J. Adobe himself, feel hemmed in by this limitation, you're not out of luck. As it turns out, you can combine font suitcases by dragging one on top of another. By doing so, you can not only organize your fonts (in font suitcases called, for example, Campbell Project or Old-Fashioned), but neatly thwart the 512-suitcase limit—because a single font suitcase can contain dozens of different fonts. (You can also, of course, buy a program like Suitcase *[www.extensis.com]* or MasterJuggler Pro *[www.alsoft.com],* which not only dodge the suitcase limit, but also let you load or unload canned sets of fonts on the fly, as you need them.)

ColorSync

As you may have read elsewhere in this book—or discovered through painful experience—computers aren't great with color. Every appliance that expresses color—such as your scanner, monitor, and printer—can "see" a different subset of the universe of color. Worse, each piece of equipment "describes" each color differently; if you've ever printed a scanned photograph on your color inkjet printer and wondered why the flesh tones, sky shades, or fruit colors didn't match the original photo, you've seen this syndrome in action. (Apple's ColorSync Web site, *www.apple.com/colorsync,* points out that "off" colors are an even bigger deal in the commercial world. A customer might return a product after discovering, for example, that the actual product color doesn't match the photo on a company's Web site.)

GEM IN THE ROUGH

The Secret Screen-Capture Keystrokes

If you're reading a chapter about printing, you may someday be interested in creating *screenshots*–printable illustrations of the Mac screen. Screenshots are a staple of articles, tutorials, and books about the Mac (including this one).

The Mac offers an enormous number of different ways to create screenshots of what you see on the screen. All of them involve pressing the ⌘ and Shift keys. Here's the rundown:

Press ⌘-Shift-3 to create a picture file on your hard drive, in the PICT graphics format, that depicts the entire screen image. A satisfying camera-shutter sound tells you that you were successful. (The file is called Picture 1. Each time you press ⌘-Shift-3, you get another file, called Picture 2, Picture 3, and so on.) You can open this file into SimpleText, Photoshop, AppleWorks, or another graphics program, in readiness for editing or printing.

Press ⌘-Shift-4 to turn your cursor into a tiny + symbol. Now drag diagonally across the screen to capture only a rectangular chunk of it. When you release the mouse, you hear the camera-click sound, and a Picture 1 file appears on your hard drive.

Add Caps Lock to the ⌘-Shift-4 keystroke to turn your cursor into a bullseye symbol. Now you can capture *only* one window or dialog box–after you click inside it. This trick saves you the trouble of cropping out unnecessary background details in your graphics program.

Add Control to either of those keystrokes if you want the resulting image to be copied onto your Clipboard, ready for pasting into (for example) Photoshop or AppleWorks, instead of creating a PICT file on your hard drive.

You can even capture a menu using these keystrokes if you first open the menu by clicking its name. (You can't capture a menu if you've dragged down it with the mouse button pressed.)

Of course, if you're really serious about capturing screenshots, opt instead for a more powerful add-on program like Snapz Pro *(www.ambrosiasw.com),* which can capture virtually anything on the screen and save it into your choice of graphics format.

Apple's ColorSync software, newly enhanced in Mac OS 9, represents a giant stride toward solving the problem. It relies on individual *profiles* for each scanner, monitor, printer, digital camera, copier, proofer, and so on—tiny files that tell the Mac how each device handles color, and which colors it can express. These profiles become part of your color documents themselves, so that every color device they encounter knows where the document came from and what it's supposed to look like. The ColorSync software then tries to supervise the color processing process, shifting colors so that they remain as consistent as possible through the scanning, editing, and printing process.

Getting ColorSync Profiles

ColorSync profiles for most Apple color printers, scanners, and monitors come built into Mac OS 9. When you buy equipment or software from Kodak, Agfa, Heidelberg, Pantone, Scitex, Imation, Barco, and Tektronix, you may get additional profiles. If your equipment didn't come with a ColorSync profile, visit Profile Central (*www.chromix.com*), where hundreds of model-specific profiles are available for downloading. (Put new profiles into the System Folder→Preferences→ColorSync Profiles folder.)

Tip: Even if your particular color appliance doesn't have a ColorSync module, you should still use the ColorSync profiles you *do* have for the other elements of your system. Every little bit helps.

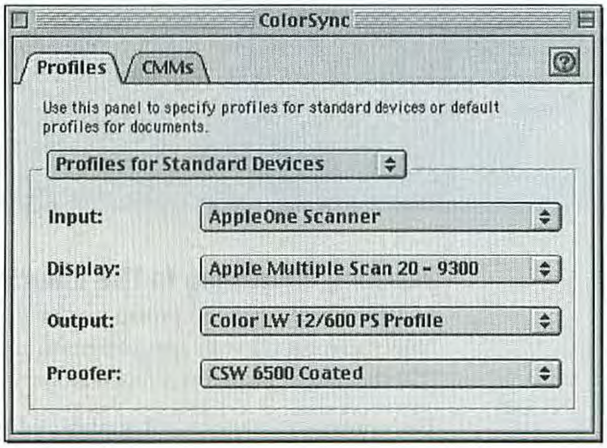
Choosing Your System's Profiles

You specify which equipment you're using by opening two control panels: Monitors and ColorSync.

Specifying your monitor's profile

Choose →Control Panels→Monitors. Click the Color button. Click the name of your monitor in the scrolling list. If you don't see it, download the list of all Apple monitor profiles from *www.apple.com/colorsync/software*. Or create your own profile by clicking Calibrate, as described on page 209.

Figure 19-10:
Tell the Macintosh what scanner and printer you intend to use. Don't bother trying to select the Display profile here; you can't adjust it. This pop-up menu merely reflects whatever choice you made in the Monitors control panel.



Use this panel to specify profiles for standard devices or default profiles for documents.

Profiles for Standard Devices

Input: AppleOne Scanner

Display: Apple Multiple Scan 20 - 9300

Output: Color LW 12/600 PS Profile

Proofer: CSW 6500 Coated

Specifying your other profiles

To indicate the scanner and printer you plan to use, choose →Control Panels→ColorSync. As you can see in Figure 19-10, you're supposed to identify the appropriate Input profile (for your scanner or camera), Output profile (for your printer), and Proofer profile (if you use a less expensive printer for checking over the work).

Saving all of this for quick switching

Fortunately, you're not condemned to using the same scanner and printer forever. Nor are you forced to switch the ColorSync control panel settings every time you switch printers or scanners.

Instead, you can save a "snapshot" of the current ColorSync control panel settings into a configuration called a ColorSync Workflow. To do so, choose File→ColorSync Workflows. A special dialog box appears, in which you can create, delete, name, or duplicate your Workflow configurations. Then, to switch from (for example) your Agfa Scanner/Kodak Printer configuration to your Kodak Camera/Proofer configuration, just click the appropriate name in this list and then click Make Active. (Using the Export and Import commands, you can also save a Workflow setup as a standalone file, suitable for transferring to other Macs so that they can duplicate your setup without your having to redo the work. And using AppleScripts, as described at the end of this chapter, you can automate Workflow switching.)

POWER USERS' CLINIC

AppleScript and ColorSync 3

Using AppleScript, described in Chapter 10, you can harness ColorSync in elaborate ways. Just by dragging document icons onto AppleScript icons, for example, you can embed ColorSync profiles, modify the already incorporated profiles, remove profiles, review the profile information embedded in a graphic, and much more. Better yet, you don't even have to know AppleScript to perform these functions–you can use the built-in AppleScripts that come with Mac OS 9.

To find them, open the Apple Extras→ColorSync Extras→AppleScript Files folder. The centerpiece of this collection of 20 ready-made AppleScripts is the Sample Scripts ReadMe, which tells you how to use them. As this document makes clear, however, Apple's real hope is that these example scripts will give you a leg up on creating your own AppleScripts. One day, Apple hopes, you, the print shop operator, will be able to automate your entire color processing routine using AppleScript and ColorSync as the centerpiece of your operation.

Teaching Photoshop to Use ColorSync

Just specifying what equipment you have isn't enough to make ColorSync work for you. Now you must tell your applications to embed ColorSync data into your documents, so that the Mac's color-correcting features will respect their color characteristics.

Each program (page layout, graphic editing, and so on) may handle ColorSync differently; some may not offer ColorSync features at all. Photoshop, for example, doesn't have ColorSync features built in—but you can add them in the form of *plug-ins.*

Because Photoshop is a core tool for most people who'd be interested in ColorSync, here's the procedure for embedding ColorSync data into your Photoshop work:

Start by downloading the ColorSync plug-ins for Photoshop from *www.apple.com/colorsync/software*. Put these three files into the Photoshop→Plug-Ins→Acquire/Export folder. Thereafter, you can ColorSync-ize an open Photoshop image by choosing Filter→Color Match→ColorSync Filter. A dialog box appears, in which you're supposed to specify the profiles for the original scanner or camera ("Source Profile") and the printer you intend to use ("Output Profile"). When you click OK, Photoshop embeds all of this information directly into the file; you're automatically on track for consistent color.

Tip: You can also use the File→Export→TIFF with ColorSync Profile command to *export* a Photoshop image with ColorSync profiles embedded in it—a great way to transfer Photoshop's familiarity with ColorSync to any program that can accept images.

CHAPTER
21

Speech Recognition and Synthesized Speech

Although it comes as a surprise to many Mac users, the Mac is quite talented when it comes to speech—especially if you're running Mac OS 9. Its abilities fall into two categories: reading text aloud, using a synthesized voice; and taking commands from your voice. This chapter shows you how to capitalize on both of these features.

FREQUENTLY ASKED QUESTION

Speech Recognition vs. Dictation Software

Can I dictate to my Mac so that it types out everything I say?

Not without add-on software. If you really want dictation software, you need a Mac program like IBM ViaVoice *(www.ibm.com)* or Voice Power Pro *(www.voicepower-pro.com)*.

By itself, Mac OS 9 still doesn't take dictation. But if your Mac has a microphone, you can speak *commands* very effectively, instructing the Mac to open and close windows and programs, click the navigation buttons in your Web browser, and so on. That's what the Mac OS 9 feature called PlainTalk Speech Recognition actually does—and that's the feature described in this chapter.

PlainTalk Speech Recognition

The Apple marketing machine may have been working too hard when it called this feature "speech recognition"—as noted in the sidebar, the Mac OS feature called PlainTalk doesn't take dictation. Instead, PlainTalk is what's known as a *command-and-control* program. It lets you open programs, trigger AppleScripts, and click menu items by speaking their names.

Few people use PlainTalk speech recognition, probably because it isn't installed as a part of the standard Mac OS 9 package, or perhaps because, until Mac OS 9, it didn't do very much. But if your Mac has a microphone, as described at the beginning of the previous chapter, PlainTalk is worth at least a 15-minute test drive; it may become a part of your work routine forever.

Installing PlainTalk Speech Recognition

To install the speech recognition software, follow these steps:

1. **Insert your Mac OS 9 CD. Double-click the Mac OS Install icon. When the welcome screen appears, click Continue.**

   Now you're asked to specify what hard drive you want to receive the installation. Almost always, the Destination Disk is your regular built-in hard drive.

2. **Click Select.**

   A small dialog box appears, letting you know that you already have Mac OS 9 installed.

3. **Click Add/Remove; on the next screen, scroll down to turn on the checkbox called English Speech Recognition. Click Start, and then click Continue.**

   When the installation is complete, you'll be asked to restart the computer; do so.

Your First Conversation with the Mac

When the Mac restarts, you'll see a few new windows on the screen (Figure 21-1). They include the Feedback window, the Speakable Commands list, and the Speech page of the Mac's online help.

The Feedback window

This is the window showing the cartoon of a young woman. The word *Esc* just below her head indicates the "listen" key—the key you're supposed to hold down when you want the Mac to respond to your voice. (You wouldn't want the Mac listening all the time—especially when you said, for example, "Hey, it's cold in here. *Close the window.*" Therefore, the Mac comes ready to listen to you only when you're pressing that key.)

You can specify a different key, if you wish, or eliminate the requirement to press a key altogether, as described in the next section.

When you start talking, you'll also see the Mac's interpretation of what you said written out here.

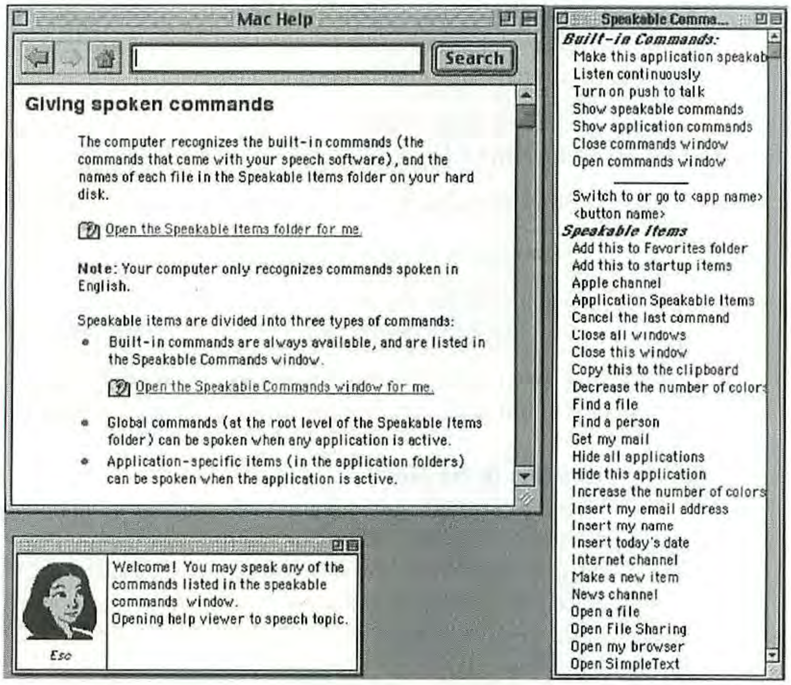
Tip: The Feedback window doesn't need to occupy as much space as it does. By clicking its Zoom box, you can collapse it down to just the cartoon head, saving a lot of horizontal space. You can also click its collapse box to shrink it vertically.

If you like having the visual feedback, though, you might simply prefer a less space-consuming "character" to appear here. The secret is to open the Speech control panel, choose Options→Feedback, and choose Character→Lights. The Lights "character" has no face at all, and makes the Feedback window very small.

The Speakable Commands window

Here's the most important single fact to understand about PlainTalk speech recognition: the only commands it understands are listed either in the Speakable Commands window, which you can't edit, or in the ⌘→Speakable Items submenu, which you *can* edit. (More on making up your own commands in a moment.)

Figure 21-1: At bottom left: the Feedback window. At right: the list of things you can say. (If these windows don't open automatically after you install PlainTalk, choose ⌘→Control Panels→Speech; choose Speakable Items from the Options pop-up menu, and click On.) At top: the Help page for the speech-recognition feature, which opens automatically.



Giving spoken commands

The computer recognizes the built-in commands (the commands that came with your speech software), and the names of each file in the Speakable Items folder on your hard disk.

Open the Speakable Items folder for me.

Note: Your computer only recognizes commands spoken in English.

Speakable items are divided into three types of commands:

- Built-in commands are always available, and are listed in the Speakable Commands window.

  Open the Speakable Commands window for me.

- Global commands (at the root level of the Speakable Items folder) can be spoken when any application is active.
- Application-specific items (in the application folders) can be spoken when the application is active.

Welcome! You may speak any of the commands listed in the speakable commands window.
Opening help viewer to speech topic.

Built-in Commands:
Make this application speakab
Listen continuously
Turn on push to talk
Show speakable commands
Show application commands
Close commands window
Open commands window

Switch to or go to <app name> <button name>

Speakable Items
Add this to Favorites folder
Add this to startup items
Apple channel
Application Speakable Items
Cancel the last command
Close all windows
Close this window
Copy this to the clipboard
Decrease the number of color
Find a file
Find a person
Get my mail
Hide all applications
Hide this application
Increase the number of colors
Insert my email address
Insert my name
Insert today's date
Internet channel
Make a new item
News channel
Open a file
Open File Sharing
Open my browser
Open SimpleText

Especially when you start using PlainTalk, the Speakable Commands window is handy: it offers a complete list of every *built-in* command (that is, those not represented by icons in the Speakable Items folder, described in this chapter). As you can see, some of them are extremely useful, and represent shortcuts that would take several steps if you had to perform them manually. Here are a few examples:

- **Switch to AppleWorks.** Here's one of the best uses of the speech-recognition program: you can launch, or switch to, one of your programs, just by using this command. (AppleWorks is just one example.)

Tip: The Mac can open two kinds of programs using this command: those listed in your ⌘→Recent Applications submenu, and those in your Speakable Items folder, as described below.

- **Add this to startup items.** Puts an alias of the highlighted icon into your System Folder→Startup Items folder, as described on page 183.

- **Close all windows.** Closes every desktop window instantly.
- **Take a window picture.** Turns your cursor into a small + symbol, so that you can drag across a rectangular area of your screen. When you release the mouse, the Mac creates a graphics file on your hard drive called Picture 1. As described in Chapter 19, this trick is useful when, for example, you want to illustrate a computer book.
- **Insert my email address.** Works in whatever program you're using. (The Mac gets your email address from the information you provided in the Internet control panel.)
- **Quit all applications.** Saves you the trouble of switching into each program and choosing File→Quit.
- **Show me what to say.** Opens the Speakable Commands window.
- **What day is it?** Tells you the date.
- **Tell me a joke.** Begins a pathetic/funny knock-knock joke. You've got to play along, providing the "who's there?" and "so-and-so *who?*" answers.

The commands listed in this window aren't the only ones you can speak, of course. You'll find out how to create new commands of your own later in this section.

Speaking to the Mac

When you decide you're ready to try talking to your computer, position the microphone. If it's the gray PlainTalk microphone that comes with Power Macs, perch it on the top of your monitor, with the Apple logo facing you. (Apple recommends that you place it between one and three feet from your mouth.) If it's a headset microphone, make sure it's plugged in. If your Mac has a built-in microphone, such as the iMac or PowerBook, you can use PlainTalk, but its recognition may not be as accurate.

In any case, finish up by opening the →Control Panels→Speech control panel; choose Options→Listening, and use the Microphone pop-up menu to specify which microphone you'll be using (if you have a choice).

Now you're ready to begin. While pressing the Esc key (if that's still the one identified in the Feedback window), begin speaking. Just speak normally and clearly; don't exaggerate or shout. Try one of the commands in the Speakable Commands list, for example—perhaps "What time is it?" If the Mac doesn't understand you, try a couple of more times, and try slowing down. (And if the Feedback window doesn't show animated sound waves, indicating that the Mac is hearing you, something's wrong with your microphone setup. Open the →Control Panels→Sound control panel again, and confirm that the Input source—your microphone—is selected correctly.)

Customizing Speech Recognition

You can tailor the speech-recognition feature in two ways—by adjusting the way it looks and operates, and by adding new commands to its vocabulary.

Changing when the Mac listens

Early experimentation quickly showed Apple's speech engineers that having the microphone "open," listening full-time, was an invitation for disaster. Everyday phone conversations, office chatter, and throat-clearings completely bewildered the software, triggering random commands or puzzled expressions from the little cartoon character.

Therefore, you must explicitly *tell* the Mac when you're addressing it. When you first install the speech-recognition feature, the Mac expects you to get its attention by pressing a key when you speak, such as the Esc key at the upper-left corner of your keyboard.

Tip: You can change the key you hold down when you want the Mac to listen. To do so, choose →Control Panels→Speech; choose Options→Listening. Click in the Key(s) field, and then press the keyboard key you'd prefer to use. Your choices are Esc, tilde (~), Delete, F5 through F15, or the keys on your numeric keypad–with or without one or more of the Shift, Control, or Option keys.

If you'd rather not have to press some key whenever you want the computer's attention, click the other option in this box, "Key(s) toggle listening on and off" (see Figure 21-2). Now you must get the computer's attention by speaking its name—which you type into the Name box—before each command. For example, you might say, "Computer, open AppleWorks," or "Hal, what day is it?" (The "push to talk" key, in this case, serves as a master on/off switch for the Mac's listening mode.)

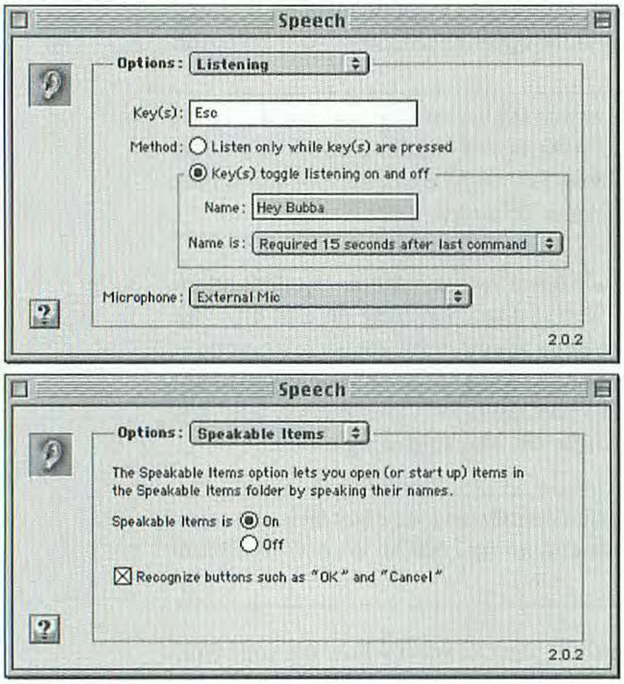
Note: This method of getting the computer's attention is generally less reliable than the push-a-key-to-talk system. *Especially* if you name the computer Hal; although that's hilarious in theory, multisyllable names work better in practice.

Using the "Name is" pop-up menu, meanwhile, you can specify how big your window of opportunity is:

- **Before every command.** When this option is selected, nothing you say is interpreted as a command unless you say the computer's name first, as in, "*Macintosh,* switch to Microsoft Word."
- **15 seconds after last command, 30 seconds after last command.** Apple offers these options for those occasions when you want to issue several commands in a row, and would feel foolish saying, "Computer, close all windows. Computer, empty the trash. Computer, switch to AppleWorks." When you turn on this option, you can say the computer's name just once; all commands that you issue in the next 15 or 30 seconds "belong to" that first salutation.

Tip: If you're not using the push-to-talk method, you can still turn speech recognition off temporarily by saying, "Turn on push to talk." (Now the Mac listens to you only when you're pressing the designated key. Your Feedback cartoon character, meanwhile, makes it clear that the Mac isn't really paying attention to you–by sleeping, reading the paper, daydreaming, and so on.) When you want to return to listening-all-the-time mode, say, "Listen continuously."

Finally, note that the Speech control panel also lets you turn off PlainTalk recognition completely. From the Options pop-up menu, choose Speakable Items, and then click Off (see Figure 21-2). (Turning off PlainTalk doesn't affect other speech features of the Mac, including voice passwords and the text-to-speech feature described at the end of this chapter.)


Speech

Options: Listening

Key(s): Esc

Method: ○ Listen only while key(s) are pressed
[✓ ● Key(s) toggle listening on and off]

Name: Hey Bubba

Name is: Required 15 seconds after last command

Microphone: External Mic

2.0.2

Speech

Options: Speakable Items

The Speakable Items option lets you open (or start up) items in the Speakable Items folder by speaking their names.

Speakable Items is [✓ ● On]
○ Off

☒ Recognize buttons such as "OK" and "Cancel"

2.0.2

Figure 21-2:
Top: If you turn on "Key(s) toggle listening on and off," then you don't have to press a key to make the Mac listen. (Instead, pressing the designated key turns the speech-recognition feature on or off completely.) In the Name field, type the name you want the Mac to listen for as it monitors the sound from your mike. Bottom: Turn off listening altogether using this On/Off control.

Changing the Feedback character

Another set of options in the Speech control panel governs the little cartoon character in the Feedback window. Choose Options→Feedback to see these choices.

- **Character.** You have a choice of nine different cartoon characters, each with basic animations, and most with an obvious lineage to popular TV. There's Connie (Chung), Sally (Jessy Raphael), Raymond (Dustin Hoffman in *Rain Man*), Phil (Hartman)—and Vincent (van Gogh), the severed ear—for example. (See Figure 21-3 for some illustrations.)

- **Speak text feedback.** Sometimes the Feedback window shows you a message of its own—when you use the "Empty the Trash" command, for example, text in the Feedback window may inform you that a locked item prevents the emptying. The Mac generally reads this text aloud to you; turn this checkbox off if you'd rather have the Mac be silent. (You can specify which of the Mac's 18 voices you want your character to use, too, by choosing Options→Voice in the Speech control panel. There's nothing to stop you from associating a male voice with a female Feedback character, and so on.)

- **Recognized.** The Mac generally makes a sound whenever it recognizes something you've said. Use this pop-up menu to control which of your built-in beeps you want it to use—or choose None.

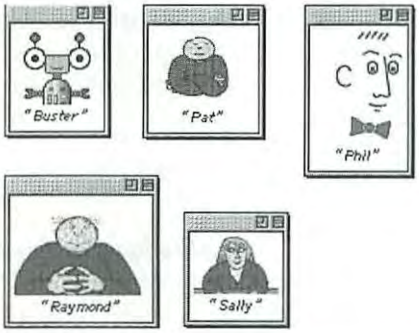


Figure 21-3:
Some say Apple's programmers have been heavily influenced by "Saturday Night Live" and other pop TV shows. How can you look at Pat, for example, without seeing the androgynous character from the SNL skit?

Improving the PlainTalk vocabulary

As you'll soon discover, PlainTalk has an extremely limited vocabulary—in fact, in addition to the canned set of Speakable Commands, it understands *only* the names of the icons in your System Folder→Apple Menu Items→Speakable Items folder. When it comes to offering you an enhanced vocabulary, the software really can't do anything more than double-click an icon—such as a document, program, folder, or alias—for you.

Tip: Actually, PlainTalk can do *one* other thing for you–it can click button names like OK, Yes, No, Quit, and Cancel. To turn on this feature, choose →Control Panels→Speech. In the Speech control panel, choose Options→Speakable Items. Turn on the checkbox called "Recognized buttons" (see Figure 21-2).

At first, you might imagine that this limitation means that PlainTalk can do little more than open programs or documents—"Open AppleWorks," "Open Internet Explorer," and so on. And indeed, that's one of PlainTalk's primary functions. By putting an alias of the favorite document or program into the Speakable Items folder, you've just taught PlainTalk to recognize its name, and to open it for you when you so command. (You can name these icons anything you want; you can also rename the starter set that Apple provides.)

Tip: PlainTalk can do the dirty work of putting favorite icons' aliases into the Speakable Items folder for you. Just highlight an icon in a desktop window and then say, "Make this speakable."

Although PlainTalk commands can't do much more than double-click icons, AppleScript icons are among them—a fact that dramatically expands PlainTalk's repertoire. (See Chapter 10 for instructions on using AppleScript.) If you choose →Speakable Items, you'll discover that most of the built-in speakable-item icons are, in fact, AppleScript icons (which look like little scrolls). As it turns out, all of the other ones—displaying a little ear—are based on AppleScripts, too. The point is that you can make PlainTalk do almost anything you want, especially in the Finder, simply by creating AppleScripts and putting them into the Speakable Items folder.

Tip: With a little bit of add-on software, PlainTalk can pull down menus for you, and type out predefined sentences or paragraphs of canned text. You could say, for example, "sign this" to have PlainTalk type out *Yours very sincerely, Jacob C. McGillicuddy, DDS.*

To make all this possible, you need a free PlainTalk add-on called ListenDo. It's available for download from *www.macspeech.com.*

Application-specific commands

Most of the pre-installed PlainTalk commands work in any program. You can say, for example, "Find a file" to launch Sherlock from within any program.

In Mac OS 9, however, you can create commands that work only in a specific program. As proof, look in your Speakable Items folder—there you'll find a folder called Application Speakable Items. And inside *that* folder are individual folders for each of the programs you might use—Internet Explorer, Outlook Express, the Finder, and so on. These commands work only when you're using those particular programs. (If you can't, or don't like to, use your hands when Web surfing, for example, you might enjoy the pre-defined browser commands like Go Back, Go Forward, Page Down, and so on.)

If you get good at AppleScript, you can create your own application-command folders in the Speakable Items→Application Speakable Items folder. Follow these steps:

1. **Launch the program for which you want to create special commands.**

   Make sure PlainTalk is on and listening.

2. **Say, "Make this application speakable."**

   The Mac creates a folder for the program in the Speakable Items folder.

3. **Drag the AppleScripts you've created into the newly created Speakable Items→ application folder.**

Of course, not every program is equally suitable to being voice-controlled—since PlainTalk is based on AppleScript, only programs that are AppleScriptable (see Chapter 10) thrive with this treatment.

Tip: If you give an application-specific icon the exact same name as one of the global commands, the Mac executes the application-specific one–if that program is running.

PlainTalk tips, tricks, and troubleshooting

When you're creating new commands, keep this advice in mind:

- The Mac understands longer icon names better than shorter ones. "Save the file" works better than "Save."
- If the name of an icon includes an acronym (such as *FTP)*, put spaces between the letters (F T P), if you'll be pronouncing them as individual letters.
- PlainTalk ignores any digits and punctuation in the Speakable Items icons' names. To open the Date & Time control panel, for example, you can say either "Open Date and Time" or, if you're in a hurry, "Open Date Time."
- You can precede the name of something in your Speakable Items folder with the word "Open." PlainTalk doesn't care—"open Excel" and "Excel" do the same thing.
- PlainTalk treats "this " and "these" identically (as in, "Add this to startup items").

If you can't seem to make the speech recognition feature work, consider this checklist:

- If your Feedback cartoon character doesn't emit sound-wave lines when you're speaking, something's wrong with your microphone arrangement. Revisit the Speech control panel, and make sure you've selected the correct microphone. Also make sure you've plugged the microphone into the correct jack on the back or side of the computer.
- Make sure you're pressing the correct key (if you're using the push-to-talk method), or speaking the name of the computer before each command (if not).
- Make sure you're saying the name of the working command, as listed either in the Speakable Commands window (which appears when you say "What can I say?") or the →Speakable Items list.
- Be aware of what program you're in. Remember that application-specific commands don't work when you're not in those programs.

PlainTalk Text-to-Speech

So far in this chapter, you've read about the Mac's listening ability. But the conversation doesn't have to be one-way. It's even easier to make the Mac talk.

In fact, the Mac can read almost anything on your screen, using your choice of 18 synthesizer voices. You hear it—the Mac speaks with a twangy, charmingly Norwegian accent—coming out of your speaker, reading whatever is on the screen in SimpleText, AppleWorks, America Online, Microsoft Word, FileMaker Pro, and several other programs.

Unlike the speech recognition feature, this talent doesn't require you to install anything special; the software is part of the standard Mac OS 9 installation. In fact, you may remember having been startled by the Mac's voice the very first time a dialog box appeared on the screen in Mac OS 9—this version of the operating system comes set to read these dialog boxes aloud, in order to get your attention.

APPENDIX U

Universal Plug and Play Device Architecture

Version 1.0, 08 Jun 2000 10:41 AM .

Table of contents

Introduction
0. Addressing
1. Discovery
2. Description
3. Control
4. Eventing
5. Presentation
Glossary

Introduction

What is Universal Plug and Play?

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

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

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

What is "universal" about UPnP? No device drivers; common protocols are used instead. UPnP networking is media independent. UPnP devices can be implemented using any programming language, and on any operating system. UPnP does not specify or constrain the design of an API for applications running on control points; OS vendors may create APIs that suit their customer's needs. UPnP enables vendor control over device UI and interaction using the browser as well as conventional application programmatic control.

ROKU EXH. 1002

UPnP Forum

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

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

UPnP vendors can build compliant devices with confidence of interoperability and benefits of shared intellectual property and the logo program. Separate from the logo program, vendors may also build devices that adhere to the UPnP Device Architecture defined herein without a formal standards procedure. If vendors build non-standard devices, they determine technical decisions that would otherwise be determined by a UPnP Forum working committee.

In this document

The Universal Plug and Play (UPnP) Device Architecture (formerly known as the DCP Framework) contained herein defines the protocols for communication between controllers, or *control points*, and devices. For discovery, description, control, eventing, and presentation, UPnP uses the following protocol stack.

UPnP vendor [purple]							
UPnP Forum [red]							
UPnP Device Architecture [green]							
HTTPMU (multicast) [black]	GENA [navy]	SSDP [blue]	HTTPU (unicast) [black]	SSDP [blue]	SOAP [blue]	HTTP [black]	GENA [navy]
					HTTP [black]		
UDP [black]					TCP [black]		
IP [black]							

At the highest layer, messages logically contain only UPnP vendor-specific information about their devices. Moving down the stack, vendor content is supplemented by information defined by UPnP Forum working committees. Messages from the layers above are hosted in UPnP-specific protocols, defined in this document. In turn, the above messages are formatted using the Simple Service Discovery Protocol (SSDP), General Event Notification Architecture (GENA), and Simple Object Access Protocol (SOAP). The above messages are delivered via HTTP, either a multicast or unicast variety running over UDP, or the standard HTTP running over TCP. Ultimately, all messages above are delivered over IP. The remaining sections of this document describe the content and format for each of these protocol layers in detail. For reference, colors in [square brackets] above indicate which protocol defines specific message components throughout this document.

The foundation for UPnP networking is IP addressing. Each device must have a Dynamic Host Configuration Protocol (DHCP) client and search for a DHCP server when the device is first connected to the network. If a DHCP server is available, i.e., the network is managed, the device must use the IP addressed assigned to it. If no DHCP server is available, i.e., the network is unmanaged, the device must use Auto IP to get an address. In brief, Auto IP defines how a device intelligently chooses an IP address from a set of reserved addresses and is able to move easily between managed and unmanaged networks. If during the DHCP transaction, the device

ROKU EXH. 1002

obtains a domain name, e.g., through a DNS server or via DNS forwarding, the device should use that name in subsequent network operations; otherwise, the device should use its IP address.

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

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

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

Step 4 in UPnP networking is eventing. A UPnP description for a service includes a list of actions the service responds to and a list of variables that model the state of the service at run time. The service publishes updates when these variables change, and a control point may subscribe to receive this information. The service publishes updates by sending event messages. Event messages contain the names of one of more state variables and the current value of those variables. These messages are also expressed in XML and formatted using the General Event Notification Architecture (GENA). A special initial event message is sent when a control point first subscribes; this event message contains the names and values for all evented variables and allows the subscriber to initialize its model of the state of the service. To support scenarios with multiple control points, eventing is designed to keep all control points equally informed about the effects of any action. Therefore, all subscribers are sent all event messages, subscribers receive event messages for all evented variables that have changed, and event messages are sent no matter why the state variable changed (either in response to a requested action or because the state the service is modeling changed). The section on Eventing below explains subscription and the format of event messages.

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

Audience

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

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

Required vs. recommended

In this document, features are described as Required, Recommended, or Optional as follows:

Required (or Must).
: These basic features must be implemented to comply with UPnP.

Recommended (or Should).
: These features add functionality supported by UPnP and should be implemented. Recommended features take advantage of the capabilities UPnP, usually without imposing major cost increases. Notice that for compliance testing, if a recommended feature is implemented, it must meet the specified requirements to be in compliance with these guidelines. Some recommended features could become requirements in the future.

Optional (or May).
: These features are neither required nor recommended by UPnP, but if the feature is implemented, it must meet the specified requirements to be in compliance with these guidelines. These features are not likely to become requirements in the future.

Acronyms

Acronym	Meaning	Acronym	Meaning
ARP	Address Resolution Protocol	SSDP	Simple Service Discovery Protocol
DHCP	Dynamic Host Configuration Protocol	UPC	Universal Product Code
DNS	Domain Name System	UPnP	Universal Plug and Play
FXPP	Flexible XML Processing Profile	URI	Uniform Resource Identifier
GENA	General Event Notification Architecture	URL	Uniform Resource Locator
HTML	HyperText Markup Language	URN	Uniform Resource Name
HTTPMU	HTTP Multicast over UDP	UUID	Universally Unique Identifier
HTTPU	HTTP (unicast) over UDP	XML	Extensible Markup Language
ICANN	Internet Corporation for Assigned Names and Numbers		
SOAP	Simple Object Access Protocol		

References and resources

RFC 2616
: HTTP: Hypertext Transfer Protocol 1.1. IETF request for comments. <http://search.ietf.org/rfc/rfc2616.txt?number=2616>.

RFC 2279

ROKU EXH. 1002

UTF-8, a transformation format of ISO 10646 (character encoding). IETF request for comments. <http://search.ietf.org/rfc/rfc2279.txt?number=2279>.

XML

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

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

Acknowledgments

The UPnP team at Microsoft gratefully acknowledges the help we received from the many technical reviewers representing the UPnP Forum Steering Committee and the UPnP vendor community. These reviewers contributed technical information, insights, and wisdom in developing this document.

We are also grateful to the software engineers, testers, and program managers at Microsoft who contributed feedback and technical content to ensure that the information in this document is accurate and timely.

0. Addressing

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

The foundation for UPnP networking is IP addressing. Each device must have a Dynamic Host Configuration Protocol (DHCP) client and search for a DHCP server when the device is first connected to the network. If a DHCP server is available, i.e., the network is managed, the device must use the IP addressed assigned to it. If no DHCP server is available, i.e., the network is unmanaged; the device must use automatic IP addressing (Auto-IP) to obtain an address.

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

The operations described in this section are further clarified in the reference documents listed below. Where conflicts between this document and the reference documents exist, the reference document always takes precedence.

0.1 Addressing: Determining whether to use Auto-IP

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

0.2 Addressing: Choosing an address

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

The selected address must then be tested to determine if the address is already in use. If the address is in use by another device, another address must be chosen and tested, up to an implementation dependent number of retries. The address selection should be randomized to avoid collision when multiple devices are attempting to allocate addresses.

0.3 Addressing: Testing the address

To test the chosen address, the device must use an Address Resolution Protocol (ARP) probe. An ARP probe is an ARP request with the device hardware address used as the sender's hardware address and the sender's IP address set to 0s. The device will then listen for responses to the ARP probe, or other ARP probes for the same IP address. If either of these ARP packets is seen, the device must consider the address in use and try a new address.

0.4 Addressing: Periodic checking for dynamic address availability

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

To switch over from one IP address to a new one, the device must cancel any outstanding advertisements and reissue new ones. The section on Discovery explains advertisements and their cancellations.

0.5 Addressing: Device naming and DNS interaction

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

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

0.6 Addressing: Name to IP address resolution

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

0.7 Addressing references

Auto-IP
    Automatically Choosing an IP Address in an Ad-Hoc IPv4 Network. IETF draft. <http://search.ietf.org/internet-drafts/draft-ietf-dhc-ipv4-autoconfig-05.txt>.
RFC1034

ROKU EXH. 1002

Domain Names - Concepts and Facilities. IETF request for comments. <http://search.ietf.org/rfc/rfc1034.txt?number=1034>.

RFC1035
: Domain Names - Implementation and Specification. IETF request for comments. <http://search.ietf.org/rfc/rfc1035.txt?number=1035>.

RFC 2131
: Dynamic Host Configuration Protocol. IETF request for comments. <http://search.ietf.org/rfc/rfc2131.txt?number=2131>.

RFC 2136
: Dynamic Updates in the Domain Name System. IETF request for comments. <http://search.ietf.org/rfc/rfc2136.txt?number=2136>.

Dynamic DNS Updates by DHCP Clients and Servers
: Interaction between DHCP and DNS. IETF Draft. <http://search.ietf.org/internet-drafts/draft-ietf-dhc-dhcp-dns-12.txt>.

1. Discovery

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

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

When a new device is added to the network, it multicasts a number of discovery messages advertising its embedded devices and services. Any interested control point can listen to the standard multicast address for notifications that new capabilities are available.

Similarly, when a new control point is added to the network, it multicasts a discovery message searching for interesting devices, services, or both. All devices must listen to the standard multicast address for these messages and must respond if any of their embedded devices or services match the search criteria in the discovery message.

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

When a device is removed from the network, it should multicast a number of discovery messages revoking it's earlier announcements, effectively declaring that it's embedded devices and services will not be available.

To limit network congestion, the time-to-live (TTL) of each IP packet for each multicast message must default to 4 and should be configurable.

Discovery plays an important role in the interoperability of devices and control points using different versions of UPnP networking. The UPnP Device Architecture (defined herein) is versioned with both a major and a minor version, usually written as *major.minor*, where both *major* and *minor* are integers. Advances in minor versions must be a compatible superset of earlier minor versions of the same major version. Advances in major version are not required to be supersets of earlier versions and are not guaranteed to be backward compatible. Version information is communicated in discovery and description messages. In the former, each discovery message includes the version of UPnP networking that the device supports. As a backup, the latter also includes the same information. This section explains the format of version information in discovery messages and specific requirements on discovery messages to maintain compatibility with advances in minor versions.

The standard multicast address, as well as the mechanisms for advertising, searching, and revoking, are defined by the Simple Service Discovery Protocol (SSDP). The remainder of this section explains SSDP in detail, enumerating how devices advertise and revoke their advertisements as well as how control points search and devices respond.

1.1 Discovery: Advertisement

When a device is added to the network, the UPnP discovery protocol allows that device to advertise its services to control points. It does this by multicasting discovery messages to a standard address and port. Control points listen to this port to detect when new capabilities are available on the network. To advertise the full extent of its capabilities, a device multicasts a number of discovery messages corresponding to each of its embedded devices and services. Each message contains information specific to the embedded device (or service) as well as information about its enclosing device. Messages should include duration until the advertisements expire; if the device remains available, the advertisements should be re-sent with (with new duration). If the device becomes unavailable, the device should explicitly cancel its advertisements, but if the device is unable to do this, the advertisements will expire on their own.

1.1.1 Discovery: Advertisement protocols and standards

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

UPnP vendor [purple]		
UPnP Forum [red]		
UPnP Device Architecture [green]		
HTTPMU (multicast) [black]	GENA [navy]	SSDP [blue]
HTTPMU (multicast) [black]		
UDP [black]		
IP [black]		

At the highest layer, discovery messages contain vendor-specific information, e.g., URL for the device description and device identifier. Moving down the stack, vendor content is supplemented by information from a UPnP Forum working committee, e.g., device type. Messages from the layers above are hosted in UPnP-specific protocols, defined in this document. In turn, the above messages are delivered via a multicast variant of HTTP that has been extended using General Event Notification Architecture (GENA) methods and headers and Simple Service Discovery Protocol (SSDP) headers. The HTTP messages are delivered via UDP over IP. For reference, colors in [square brackets] above indicate which protocol defines specific headers and values in discovery messages listed below.

1.1.2 Discovery: Advertisement: Device available -- NOTIFY with ssdp:alive

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

ROKU EXH. 1002

1. a potential search target (e.g., device type), sent in an NT header,
2. a composite identifier for the advertisement, sent in a USN header,
3. a URL for more information about the device (or enclosing device in the case of a service), sent in a LOCATION header, and
4. a duration for which the advertisement is valid, sent in a CACHE-CONTROL header.

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

- Three discovery messages for the root device.

	NT	USN *
1	root device UUID **	root device UUID
2	device type : device version	root device UUID and :: and device type : device version
3	upnp:rootdevice	root device UUID and :: and upnp:rootdevice

- Two discovery messages for each embedded device.

	NT	USN *
1	embedded device UUID **	embedded device UUID
2	device type : device version	embedded device UUID and :: and device type : device version

- Once for each service.

	NT	USN *
1	service type : service version	enclosing device UUID and :: and service type : service version

* Note that the prefix of the USN header (before the double colon) must match the value of the UDN element in the device description. (The section on Description explains the UDN element.)
** Note that the value of this NT header must match the value of the UDN element in the device description.

If a root device has *d* embedded devices and *s* embedded services but only *k* distinct service types, this works out to 3+2*d*+*k* requests. This advertises the full extend of the device's capabilities to interested control points. These messages must be sent out as a series with roughly comparable expiration times; order is unimportant, but refreshing or canceling individual messages is prohibited.

Choosing an appropriate duration for advertisements is a balance between minimizing network traffic and maximizing freshness of device status. Relatively short durations close to the minimum of 1800 seconds will ensure that control points have current device status at the expense of additional network traffic; longer durations, say on the order of a day, compromise freshness of device status but can significantly reduce network traffic. Generally, device vendors should choose a value that corresponds to expected device usage: short durations for devices that are expected to be part of the network for short periods of time, and significantly longer durations for devices expected to be long-term members of the network.

Due to the unreliable nature of UDP, devices should send each of the above discovery messages more than once. As a fallback, to guard against the possibility that a control point might not receive an advertisement for a device or service, the device should re-send its advertisements periodically (cf. CACHE-CONTROL below). Note that UDP packets are also bounded in length (perhaps as small as 512 Bytes in some implementations) and that there is no guarantee that the above 3+2*d*+*k* messages will arrive in a particular order.

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

```
NOTIFY * HTTP/1.1
HOST: 239.255.255.250:1900
```

ROKU EXH. 1002

```
CACHE-CONTROL: max-age = seconds until advertisement expires
LOCATION: URL for UPnP description for root device
NT: search target
NTS: ssdp:alive
SERVER: OS/version UPnP/1.0 product/version
USN: advertisement UUID
```

(No body for request with method NOTIFY, but note that the message must have a blank line following the last HTTP header.)

The TTL for the IP packet must default to 4 and should be configurable.

Listed below are details for the request line and headers appearing in the listing above. All header values are case sensitive except where noted.

Request line

NOTIFY
: Method defined by GENA for sending notifications and events.

*
: Request applies generally and not to a specific resource. Must be *.

HTTP/1.1
: HTTP version.

Headers

HOST
: Required. Multicast channel and port reserved for SSDP by Internet Assigned Numbers Authority (IANA). Must be 239.255.255.250:1900.

CACHE-CONTROL
: Required. Must have max-age directive that specifies number of seconds the advertisement is valid. After this duration, control points should assume the device (or service) is no longer available. Should be > 1800 seconds (30 minutes). Specified by UPnP vendor. Integer.

LOCATION
: Required. Contains a URL to the UPnP description of the root device. In some unmanaged networks, host of this URL may contain an IP address (versus a domain name). Specified by UPnP vendor. Single URL.

NT
: Required header defined by GENA. Notification Type. Must be one of the following. (cf. table above.) Single URI.

  upnp:rootdevice
  : Sent once for root device.

  uuid:device-UUID
  : Sent once for each device, root or embedded. Device UUID specified by UPnP vendor.

  urn:schemas-upnp-org:device:deviceType:v
  : Sent once for each device, root or embedded. Device type and version defined by UPnP Forum working committee.

  urn:schemas-upnp-org:service:serviceType:v
  : Sent once for each service. Service type and version defined by UPnP Forum working committee.

NTS
: Required header defined by GENA. Notification Sub Type. Must be ssdp:alive. Single URI.

SERVER

ROKU EXH. 1002

Required. Concatenation of OS name, OS version, UPnP/1.0, product name, and product version. Specified by UPnP vendor. String.

USN

Required header defined by SSDP. Unique Service Name. Must be one of the following. (cf. table above.) The prefix (before the double colon) must match the value of the UDN element in the device description. (The section on Description explains the UDN element.) Single URI.

uuid:device-UUID::upnp:rootdevice
: Sent once for root device. Device UUID specified by UPnP vendor.

uuid:device-UUID
: Sent once for every device, root or embedded. Device UUID specified by UPnP vendor.

uuid:device-UUID::urn:schemas-upnp-org:device:deviceType:v
: Sent once for every device, root or embedded. Device UUID specified by UPnP vendor. Device type and version defined by UPnP Forum working committee.

uuid:device-UUID::urn:schemas-upnp-org:service:serviceType:v
: Sent once for every service. Device UUID specified by UPnP vendor. Service type and version defined by UPnP Forum working committee.

(No response for a request with method NOTIFY.)

1.1.3 Discovery: Advertisement: Device unavailable -- NOTIFY with ssdp:byebye

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

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

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

```
NOTIFY * HTTP/1.1
HOST: 239.255.255.250:1900
NT: search target
NTS: ssdp:byebye
USN: advertisement UUID
```

(No body for request with method NOTIFY, but note that the message must have a blank line following the last HTTP header.)

The TTL for the IP packet must default to 4 and should be configurable.

Listed below are details for the request line and headers appearing in the listing above. All header values are case sensitive except where noted.

Request line

NOTIFY

Method defined by GENA for sending notifications and events.

ROKU EXH. 1002

*
: Request applies generally and not to a specific resource. Must be *.

HTTP/1.1
: HTTP version.

Headers

HOST
: Required. Multicast channel and port reserved for SSDP by Internet Assigned Numbers Authority (IANA). Must be 239.255.255.250:1900.

NT
: Required header defined by GENA. Notification Type. (See list of required values for NT header in NOTIFY with ssdp:alive above.) Single URI.

NTS
: Required header defined by GENA. Notification Sub Type. Must be ssdp:byebye. Single URI.

USN
: Required header defined by SSDP. Unique Service Name. (See list of required values for USN header in NOTIFY with ssdp:alive above.) Single URI.

(No response for a request with method NOTIFY.)

Due to the unreliable nature of UDP, devices should send each of the above messages more than once. As a fallback, if a control point fails to receive notification that a device or services is unavailable, the original discovery message will eventually expire yielding the same effect.

1.2 Discovery: Search

When a control point is added to the network, the UPnP discovery protocol allows that control point to search for devices of interest on the network. It does this by multicasting a search message with a pattern, or target, equal to a type or identifier for a device or service. Responses from devices contain discovery messages essentially identical to those advertised by newly connected devices; the former are unicast while the latter are multicast.

1.2.1 Discovery: Search protocols and standards

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

UPnP vendor [purple]			
UPnP Forum [red]			
UPnP Device Architecture [green]			
HTTPU (unicast) [black]	SSDP [blue]	HTTPMU (multicast) [black]	SSDP [blue]
UDP [black]			
IP [black]			

At the highest layer, search messages contain vendor-specific information, e.g., the control point, device, and service identifiers. Moving down the stack, vendor content is supplemented by information from a UPnP Forum working committee, e.g., device or service types. Messages from the layers above are hosted in UPnP-specific protocols, defined in this document. In turn, search requests are delivered via a multicast variant of HTTP that has been extended using Simple Service Discovery Protocol (SSDP) methods headers. Search responses are delivered via a unicast variant of HTTP that has also been extended with SSDP. (GENA is not involved when control points search for devices.) Both kinds of HTTP messages are delivered via UDP over IP. For reference, colors in [square brackets] above indicate which protocol defines specific headers and values in discovery messages listed below.

1.2.2 Discovery: Search: Request with M-SEARCH

When a control point is added to the network, it should send a multicast request with method M-SEARCH in the following format. Values in *italics* are placeholders for actual values.

```
M-SEARCH * HTTP/1.1
HOST: 239.255.255.250:1900
MAN: "ssdp:discover"
MX: seconds to delay response
ST: search target
```

(No body for request with method M-SEARCH, but note that the message must have a blank line following the last HTTP header.)

The TTL for the IP packet must default to 4 and should be configurable.

Listed below are details for the request line and headers appearing in the listing above. All header values are case sensitive except where noted.

Request line

M-SEARCH
: Method defined by SSDP for search requests.

*
: Request applies generally and not to a specific resource. Must be *.

HTTP/1.1
: HTTP version.

Headers

HOST
: Required. Multicast channel and port reserved for SSDP by Internet Assigned Numbers Authority (IANA). Must be 239.255.255.250:1900.

MAN
: Required. Unlike the NTS and ST headers, the value of the MAN header is enclosed in double quotes. Must be "ssdp:discover".

MX
: Required. Maximum wait. Device responses should be delayed a random duration between 0 and this many seconds to balance load for the control point when it processes responses. This value should be increased if a large number of devices are expected to respond or if network latencies are expected to be significant. Specified by UPnP vendor. Integer.

ST
: Required header defined by SSDP. Search Target. Must be one of the following. (cf. NT header in NOTIFY with ssdp:alive above.) Single URI.

    ssdp:all
    : Search for all devices and services.

ROKU EXH. 1002

upnp:rootdevice
: Search for root devices only.

uuid:device-UUID
: Search for a particular device. Device UUID specified by UPnP vendor.

urn:schemas-upnp-org:device:deviceType:v
: Search for any device of this type. Device type and version defined by UPnP Forum working committee.

urn:schemas-upnp-org:service:serviceType:v
: Search for any service of this type. Service type and version defined by UPnP Forum working committee.

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

1.2.3 Discovery: Search: Response

To be found, a device must send a response to the source IP address and port that sent the request to the multicast channel.

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

```
HTTP/1.1 200 OK
CACHE-CONTROL: max-age = seconds until advertisement expires
DATE: when response was generated
EXT:
LOCATION: URL for UPnP description for root device
SERVER: OS/version UPnP/1.0 product/version
ST: search target
USN: advertisement UUID
```

(No body for a response to a request with method M-SEARCH, but note that the message must have a blank line following the last HTTP header.)

(No need to limit TTL to 4 for the IP packet in response to a search request.)

Listed below are details for the headers appearing in the listing above. All header values are case sensitive except where noted.

Headers

CACHE-CONTROL
: Required. Must have max-age directive that specifies number of seconds the advertisement is valid. After this duration, control points should assume the device (or service) is no longer available. Should be > 1800 seconds (30 minutes). Specified by UPnP vendor. Integer.

DATE
: Recommended. When response was generated. RFC 1123 date.

EXT
: Required. Confirms that the MAN header was understood. (Header only; no value.)

LOCATION

Required. Contains a URL to the UPnP description of the root device. In some unmanaged networks, host of this URL may contain an IP address (versus a domain name). Specified by UPnP vendor. Single URL.

SERVER

Required. Concatenation of OS name, OS version, UPnP/1.0, product name, and product version. Specified by UPnP vendor. String.

ST

Required header defined by SSDP. Search Target. Single URI. If ST header in request was,

ssdp:all

Respond 3+2*d*+*k* times for a root device with *d* embedded devices and *s* embedded services but only *k* distinct service types. Value for ST header must be the same as for the NT header in NOTIFY messages with ssdp:alive. (See above.) Single URI.

upnp:rootdevice

Respond once for root device. Must be upnp:rootdevice. Single URI.

uuid:device-UUID

Respond once for each device, root or embedded. Must be uuid:device-UUID. Device UUID specified by UPnP vendor. Single URI.

urn:schemas-upnp-org:device:deviceType:v

Respond once for each device, root or embedded. Must be urn:schemas-upnp-org:device:deviceType:v. Device type and version defined by UPnP Forum working committee.

urn:schemas-upnp-org:service:serviceType:v

Respond once for each service. Must be urn:schemas-upnp-org:service:serviceType:v. Service type and version defined by UPnP Forum working committee.

USN

Required header defined by SSDP. Unique Service Name. (See list of required values for USN header in NOTIFY with ssdp:alive above.) Single URI.

Due to the unreliable nature of UDP, devices should send each response more than once. As a fallback, to guard against the possibility that a control point not receive a response, a device should re-send its advertisements periodically (cf. CACHE-CONTROL header in NOTIFY with ssdp:alive above).

If there is an error with the search request, the device must send a response with one of the following errors.

Errors

MAN header != ssdp:discover

412 Precondition Failed. If the value of the MAN header is not equal to ssdp:discover, the device must respond with HTTP error 412 Precondition Failed.

Other errors may be returned by layers in the protocol stack below UPnP. Consult documentation on those protocols for details.

1.3 Discovery references

GENA

General Event Notification Architecture. IETF Draft.

HTTPMU

HTTPU

HTTP Multicast over UDP, HTTP Unicast over UDP. IETF Draft.

SSDP

Simple Service Discovery Protocol. IETF Draft.

2. Description

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

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

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

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

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

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

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

UPnP vendors can differentiate their devices by extending services, including additional UPnP services, or embedding additional devices. When a control point retrieves a particular device's description, these added features are exposed to the control point for control and eventing. The device and service descriptions authoritatively document the implementation of the device.

Retrieving a UPnP device description is simple: the control point issues an HTTP GET request on the URL in the discovery message, and the device returns the device description. Retrieving a UPnP service description is a similar process that uses a URL within the device description. The protocol stack, method, headers, and body for the response and request are explained in detail below.

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. The device and service descriptions may be retrieved at any point since the device and service descriptions are static as long as the device and its services are available. If a device cancels its advertisements, a control point must assume the device and its services are no longer available. If a device needs to change one of these descriptions, it must cancel its outstanding advertisements and re-advertise. Consequently, control points should not assume that device and service descriptions are unchanged if a device re-appears on the network.

Like discovery, description plays an important role in the interoperability of devices and control points using different versions of UPnP networking. As explained in the section on Discovery, The UPnP Device Architecture (defined herein) is versioned with both a major and a minor version. Advances in minor versions must be a compatible superset of earlier minor versions of the same major version. Advances in major version are not required to be supersets of earlier versions and are not guaranteed to be backward compatible. Version information is communicated in description messages as a backup to the information communicated in discovery messages. This section explains the format of version information in description messages.

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

2.1 Description: Device description

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

ROKU EXH. 1002

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

Listed below are details for each of the elements, attributes, and values appearing in the listing above. All elements and attributes are case sensitive; HTTP specifies case sensitivity for URLs; other values are not case sensitive except where noted. The order of elements is insignificant. Except where noted: required elements must occur exactly once (no duplicates), and recommended or optional elements may occur at most once.

xml

Required for all XML documents. Case sensitive.

root

Required. Must have urn:schemas-upnp-org:device-1-0 as the value for the xmlns attribute; this references the UPnP Template Language (described below). Case sensitive. Contains all other elements describing the root device, i.e., contains the following sub elements:

ROKU EXH. 1002

specVersion
: Required. Contains the following sub elements:

  major
  : Required. Major version of the UPnP Device Architecture. Must be 1.

  minor
  : Required. Minor version of the UPnP Device Architecture. Must be 0.

URLBase
: Optional. Defines the base URL. Used to construct fully-qualified URLs. All relative URLs that appear elsewhere in the description are appended to this base URL. If URLBase is empty or not given, the base URL is the URL from which the device description was retrieved. Specified by UPnP vendor. Single URL.

device
: Required. Contains the following sub elements:

  deviceType
  : Required. UPnP device type.

    - For standard devices defined by a UPnP Forum working committee, must begin with urn:schemas-upnp-org:device: followed by a device type suffix, colon, and an integer device version (as shown in the listing above).
    - For non-standard devices specified by UPnP vendors, must begin with urn:, followed by an ICANN domain name owned by the vendor, followed by :device:, followed by a device type suffix, colon, and an integer version, i.e., urn:domain-name:device:deviceType:v.

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

  friendlyName
  : Required. Short description for end user. Should be localized (cf. ACCEPT-/CONTENT-LANGUAGE headers). Specified by UPnP vendor. String. Should be < 64 characters.

  manufacturer
  : Required. Manufacturer's name. May be localized (cf. ACCEPT-/CONTENT-LANGUAGE headers). Specified by UPnP vendor. String. Should be < 64 characters.

  manufacturerURL
  : Optional. Web site for Manufacturer. May be localized (cf. ACCEPT-/CONTENT-LANGUAGE headers). May be relative to base URL. Specified by UPnP vendor. Single URL.

  modelDescription
  : Recommended. Long description for end user. Should be localized (cf. ACCEPT-/CONTENT-LANGUAGE headers). Specified by UPnP vendor. String. Should be < 128 characters.

  modelName
  : Required. Model name. May be localized (cf. ACCEPT-/CONTENT-LANGUAGE headers). Specified by UPnP vendor. String. Should be < 32 characters.

  modelNumber
  : Recommended. Model number. May be localized (cf. ACCEPT-/CONTENT-LANGUAGE headers). Specified by UPnP vendor. String. Should be < 32 characters.

  modelURL
  : Optional. Web site for model. May be localized (cf. ACCEPT-/CONTENT-LANGUAGE headers). May be relative to base URL. Specified by UPnP vendor. Single URL.

  serialNumber
  : Recommended. Serial number. May be localized (cf. ACCEPT-/CONTENT-LANGUAGE headers). Specified by UPnP vendor. String. Should be < 64 characters.

UDN

Required. Unique Device Name. Universally-unique identifier for the device, whether root or embedded. Must be the same over time for a specific device instance (i.e., must survive reboots). Must match the value of the NT header in device discovery messages. Must match the prefix of the USN header in all discovery messages. (The section on Discovery explains the NT and USN headers.) Must begin with uuid: followed by a UUID suffix specified by a UPnP vendor. Single URI.

UPC

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

iconList

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

icon

Recommended. Icon to depict device in a control point UI. May be localized (cf. ACCEPT-/CONTENT-LANGUAGE headers). Recommend one icon in each of the following sizes (width x height x depth): 16x16x1, 16x16x8, 32x32x1, 32x32x8, 48x48x1, 48x48x8. Contains the following sub elements:

mimetype

Required. Icon's MIME type (cf. RFC 2387). Single MIME image type.

width

Required. Horizontal dimension of icon in pixels. Integer.

height

Required. Vertical dimension of icon in pixels. Integer.

depth

Required. Number of color bits per pixel. Integer.

url

Required. Pointer to icon image. (XML does not support direct embedding of binary data. See note below.) Retrieved via HTTP. May be relative to base URL. Specified by UPnP vendor. Single URL.

serviceList

Required. Contains the following sub elements:

service

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

serviceType

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

- For standard service types defined by a UPnP Forum working committee, must begin with urn:schemas-upnp-org:service: followed by a service type suffix, colon, and an integer service version (as shown in the listing above).
- For non-standard service types specified by UPnP vendors, must begin with urn:, followed by an ICANN domain name owned by the vendor, followed by :service:, followed by a service type suffix, colon, and an integer service version, i.e., urn:domain-name:service:serviceType:v.

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

ROKU EXH. 1002

suffix and separating colon. Single URI.

serviceId

Required. Service identifier. Must be unique within this device description.

- For standard services defined by a UPnP Forum working committee, must begin with urn:upnp-org:serviceId: followed by a service ID suffix (as shown in the listing above). (Note that upnp-org is used instead of schemas-upnp-org in this case because an XML schema is not defined for each service ID.)
- For non-standard services specified by UPnP vendors, must begin with urn:, followed by an ICANN domain name owned by the vendor, followed by :serviceId:, followed by a service ID suffix, i.e., urn:domain-name:serviceId:serviceID.

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

SCPDURL

Required. URL for service description (nee Service Control Protocol Definition URL). (cf. section below on service description.) May be relative to base URL. Specified by UPnP vendor. Single URL.

controlURL

Required. URL for control (cf. section on Control). May be relative to base URL. Specified by UPnP vendor. Single URL.

eventSubURL

Required. URL for eventing (cf. section on Eventing). May be relative to base 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 (cf. section on Eventing); if the service does not have eventing, this element must be present but should be empty, i.e., <eventSubURL></eventSubURL>. Specified by UPnP vendor. Single URL.

deviceList

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

device

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

presentationURL

Recommended. URL to presentation for device (cf. section on Presentation). May be relative to base URL. Specified by UPnP vendor. Single URL.

For future extensibility, when processing XML like the listing above, as specified by the Flexible XML Processing Profile (FXPP), devices and control points must ignore: (a) any unknown elements and their sub elements or content, and (b) any unknown attributes and their values.

Note that ampersand character (&, 0x26 in UTF-8) is not allowed in XML. If required as part of the value of an XML element (e.g., a URL), the ampersand character must be converted into &amp; (HTML) or %26 (URL escape code).

XML does not support directly embedding binary data, e.g., icons in UPnP device descriptions. Binary data may be converted into text (and thereby embedded into XML) using an XML data type of either bin.base64 (a

ROKU EXH. 1002

MIME-style base 64 encoding for binary data) or bin.hex (hexadecimal digits represent octets). Alternatively, the data can be passed indirectly, as it were, by embedding a URL in the XML and transferring the data in response to a separate HTTP request; the icon(s) in UPnP device descriptions are transferred in this latter manner.

Devices standardized by UPnP Forum working committees have an integer version. Every later version of a device must be a superset of the previous version, i.e., compared to earlier versions of the device, it must include all embedded devices and services of the same or later version. The UPnP device type remains the same across all versions of a device whereas the device version must be larger for later versions.

2.2 Description: UPnP Device Template

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

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

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

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

2.3 Description: Service description

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

Each service may have zero or more actions. Each action may have zero or more arguments. Any combination of these arguments may be input or output parameters. If an action has one or more output arguments, one these arguments may be marked as a return value. Each argument should correspond to a state variable. This direct-manipulation programming model reinforces simplicity.

Each service must have one or more state variables.

In addition to defining non-standard services, UPnP vendors may add actions and services to standard devices.

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

```
<?xml version="1.0"?>
<scpd xmlns="urn:schemas-upnp-org:service-1-0">
  <specVersion>
    <major>1</major>
    <minor>0</minor>
  </specVersion>
  <actionList>
    <action>
      <name>actionName</name>
      <argumentList>
        <argument>
          <name>formalParameterName</name>
          <direction>in xor out</direction>
          <retval />
          <relatedStateVariable>stateVariableName</relatedStateVariable>
        </argument>
       Declarations for other arguments defined by UPnP Forum working committee (if any)
          go here
      </argumentList>
    </action>
    Declarations for other actions defined by UPnP Forum working committee (if any)
      go here
    Declarations for other actions added by UPnP vendor (if any) go here
  </actionList>
  <serviceStateTable>
    <stateVariable sendEvents="yes">
      <name>variableName</name>
      <dataType>variable data type</dataType>
      <defaultValue>default value</defaultValue>
      <allowedValueList>
        <allowedValue>enumerated value</allowedValue>
        Other allowed values defined by UPnP Forum working committee (if any) go here
      </allowedValueList>
    </stateVariable>
    <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>
    Declarations for other state variables defined by UPnP Forum working committee
      (if any) go here
    Declarations for other state variables added by UPnP vendor (if any) go here
  </serviceStateTable>
</scpd>
```

Listed below are details for each of the elements, attributes, and values appearing in the listing above. All elements and attributes are case sensitive; values are not case sensitive except where noted. Except where noted, the order of elements is insignificant. Except where noted, required elements must occur exactly once (no duplicates), and recommended or optional elements may occur at most once.

xml
: Required for all XML documents. Case sensitive.

scpd
: Required. Must have urn:schemas-upnp-org:service-1-0 as the value for the xmlns attribute; this references the UPnP Template Language (explained below). Case sensitive. Contains all other elements describing the service, i.e., contains the following sub elements:

ROKU EXH. 1002

specVersion
: Required. Contains the following sub elements:

  major
  : Required. Major version of the UPnP Device Architecture. Must be 1.

  minor
  : Required. Minor version of the UPnP Device Architecture. Must be 0.

actionList
: Required if and only if the service has actions. (Each service may have >= 0 actions.) Contains the following sub element(s):

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

    name
    : Required. Name of action. Must not contain a hyphen character (-, 2D Hex in UTF-8) nor a hash character (#, 23 Hex in UTF-8).

      - For standard actions defined by a UPnP Forum working committee, must not begin with X_ nor A_.
      - For non-standard actions specified by a UPnP vendor and added to a standard service, must begin with X_.

      String. Should be < 32 characters.

    argumentList
    : Required if and only if parameters are defined for action. (Each action may have >= 0 parameters.) Contains the following sub element(s):

      argument
      : Required. Repeat once for each parameter. Contains the following sub elements:

        name
        : Required. 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). String. Should be < 32 characters.

        direction
        : Required. Whether argument is an input or output parameter. Must be in xor out. 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.

serviceStateTable
: Required. (Each service must have > 0 state variables.) Contains the following sub element(s):

  stateVariable
  : Required. Repeat once for each state variable defined by a UPnP Forum working committee. If UPnP vendor differentiates service by adding additional state variables, repeat once for each additional variable. sendEvents attribute defines whether event messages will be

generated when the value of this state variable changes; non-evented state variables have sendEvents="no"; default is sendEvents="yes". Contains the following sub elements:

name
: Required. 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_ nor A_.
- For non-standard variables specified by a UPnP vendor and added to a standard service, must begin with X_.

String. Should be < 32 characters.

dataType
: Required. Same as data types defined by XML Schema, Part 2: Datatypes. Defined by a UPnP Forum working committee for standard state variables; specified by UPnP vendor for extensions. Must be one of the following values:

ui1
: Unsigned 1 Byte int. Same format as int without leading sign.

ui2
: Unsigned 2 Byte int. Same format as int without leading sign.

ui4
: Unsigned 4 Byte int. Same format as int without leading sign.

i1
: 1 Byte int. Same format as int.

i2
: 2 Byte int. Same format as int.

i4
: 4 Byte int. Same format as int. Must be between -2147483648 and 2147483647.

int
: Fixed point, integer number. May have leading sign. May have leading zeros. (No currency symbol.) (No grouping of digits to the left of the decimal, e.g., no commas.)

r4
: 4 Byte float. Same format as float. Must be between 3.40282347E+38 to 1.17549435E-38.

r8
: 8 Byte float. Same format as float. Must be between -1.79769313486232E308 and -4.94065645841247E-324 for negative values, and between 4.94065645841247E-324 and 1.79769313486232E308 for positive values, i.e., IEEE 64-bit (8-Byte) double.

number
: Same as r8.

fixed.14.4
: Same as r8 but no more than 14 digits to the left of the decimal point and no more than 4 to the right.

float
: Floating point number. Mantissa (left of the decimal) and/or exponent may have a leading sign. Mantissa and/or exponent may have leading zeros. Decimal character in mantissa is a period, i.e., whole digits in mantissa separated from fractional digits by period. Mantissa separated from exponent by E. (No currency symbol.) (No grouping of digits in the mantissa, e.g., no commas.)

char

ROKU EXH. 1002

Unicode string. One character long.

string

Unicode string. No limit on length.

date

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

dateTime

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

dateTime.tz

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

time

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

time.tz

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

boolean

0, false, or no for false; 1, true, or yes for true.

bin.base64

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

bin.hex

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

uri

Universal Resource Identifier.

uuid

Universally Unique ID. Hexadecimal digits representing octets. Optional embedded hyphens are ignored.

defaultValue

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.

allowedValueList

Recommended. Enumerates legal string values. Prohibited for data types other than string. At most one of allowedValueRange and allowedValueList may be specified. Sub elements are ordered (e.g., see NEXT_STRING_BOUNDED). Contains the following sub elements:

allowedValue

Required. A legal value for a string variable. Defined by a UPnP Forum working committee for standard state variables; specified by UPnP vendor for extensions. string. Should be < 32 characters.

allowedValueRange

Recommended. Defines bounds for legal numeric values; defines resolution for numeric values. Defined only for numeric data types. At most one of allowedValueRange and allowedValueList may be specified. Contains the following sub elements:

minimum

Required. Inclusive lower bound. Defined by a UPnP Forum working committee or delegated to UPnP vendor. Single numeric value.

maximum

ROKU EXH. 1002

Required. Inclusive upper bound. Defined by a UPnP Forum working committee or delegated to UPnP vendor. Single numeric value.

step

Recommended. Size of an increment operation, i.e., value of *s* in the operation *v* = *v* + *s*. Defined by a UPnP Forum working committee or delegated to UPnP vendor. Single numeric value.

For future extensibility, when processing XML like the listing above, as specified by the Flexible XML Processing Profile (FXPP), devices and control points must ignore: (a) any unknown elements and their sub elements or content, and (b) any unknown attributes and their values.

Note that ampersand character (&, 26 Hex in UTF-8) is not allowed in XML. If required as part of the value of an XML element (e.g., a URL), the ampersand character must be converted into &amp; (HTML) or %26 (URL escape code).

Note that it is logically possible for a service to have no actions but have state variables and eventing; though unlikely, such a service would be an autonomous information source. However, a service with no state variables is prohibited.

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

However, there may be some cases where an action's behavior is locale-dependent. In this case, an argument should be defined to indicate the locale, perhaps using the same encoding as the ACCEPT-/CONTENT-LANGUAGE headers (RFC 1766). If there are multiple locale-dependent actions, the service may include an action to set a state variable to indicate the locale and eliminate the need to pass a locale identifier separately to each action.

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

2.4 Description: UPnP Service Template

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

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

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

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

2.5 Description: Non-standard vendor extensions

As explained above, UPnP vendors may differentiate their devices and extend a standard device by including additional services, embedded devices. Similarly, UPnP vendors may extend a standard service by including additional actions or state variables. UPnP vendors must not extend a standard service by modifying a standardized allowedValueList. Naming conventions for each of these are listed in the table below and explained in detail above.

Type of extension	Standard	Non-Standard
device type	urn:schemas-upnp-org:device:deviceType:v	urn:domain-name:device:deviceType:v
service type	urn:schemas-upnp-org:service:serviceType:v	urn:domain-name:service:serviceType:v
service ID	urn:upnp-org:serviceId:serviceID	urn:domain-name:serviceId:serviceID
action name	Does not begin with X_ or A_.	Begins with X_.
state variable name	Does not begin with X_ or A_.	Begins with X_.
XML elements in device or service description	Defined by the UPnP Template Language.	Arbitrary XML scoped by an XML namespace and nested within an element that begins with X_.
XML attributes in device or service description	Defined by the UPnP Template Language.	Arbitrary attributes scoped by an XML namespace and begin with X_.

As the last two rows of the table above indicate, UPnP vendors may also add non-standard XML to a device or service description. Each addition must be scoped by a vendor-supplied XML namespace. Arbitrary XML must be enclosed in an element that begins with X_, and this element must be a sub element of standard element that contains sub elements. Non-standard attributes may be added to standard elements provided these attributes are scoped by an XML namespace and begin with X_.

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

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

ROKU EXH. 1002

RootStandardElement
Required. A standard root element. xmlns attribute defines namespaces, in this case, a standard UPnP namespace and a non-standard namespace with the prefix n.

- For device descriptions, must be root.
- For service descriptions, must be scpd.

AnyStandardElement
Required. Any standard element, root or otherwise, content of text or element only. Must already be included as part of the standard device or service description. X_VendorAttribute must begin with X_. (Prefix A_ is reserved.) May have an arbitrary string value.

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

EltOnlyStandardElement
Required. Element with content of element only. Must already be included as part of the standard device or service description.

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

X_VendorElement
Required. Must begin with X_. (Prefix A_ is reserved.) Must have a value for the xmlns attribute. May contain arbitrary XML.

As specified by the Flexible XML Processing Profile (FXPP), control points that do not understand these XML additions must ignore them.

2.6 Description: UPnP Template Language for devices

The paragraphs above explain UPnP device descriptions and illustrate how one would be instantiated from a UPnP Device Template. As explained, UPnP Device Templates are produced by UPnP Forum working committees, and these templates are derived from the UPnP Template Language. This template language defines valid templates for devices and services. Below is a listing and explanation of this language as it pertains to devices.

The UPnP Template Language is written in XML syntax and is derived from XML Schema (Part 1: Structures, Part 2: Datatypes). XML Schema provides a set of XML constructions that express language concepts like required vs. optional elements, element nesting, and data types for values (as well as other properties not of interest here). The UPnP Template Language uses these XML Schema constructions to define elements like specVersion, URLBase, deviceType, et al listed in detail above. Because the UPnP Template Language is constructed using another, precise language, it is unambiguous. And because the UPnP Template Language, UPnP Device Templates, and UPnP device descriptions are all machine-readable, automated tools can automatically check to ensure the latter two have all required elements, are correctly nested, and have values of the correct data types.

Below is the UPnP Template Language for devices as defined by the UPnP Device Architecture herein. The elements it defines are used in UPnP Device Templates; they are colored green here, and they are colored green in the listing above. Below is where these elements are defined; above is where they are used.

ROKU EXH. 1002

Immediately following this is a brief explanation of the XML Schema elements, attributes, and values used. The reference to XML Schema at the end of the section has further details.

UPnP Template Language for devices

```
<?xml version="1.0"?>
<Schema name="device-1-0"
    xmlns="urn:schemas-microsoft-com:xml-data"
    xmlns:dt="urn:schemas-microsoft-com:datatypes">
  <ElementType name="root" content="eltOnly">
    <element type="specVersion" />
    <element type="URLBase" minOccurs="0" maxOccurs="1" />
    <element type="device" />
  </ElementType>
  <ElementType name="specVersion" content="eltOnly">
    <element type="major" />
    <element type="minor" />
  </ElementType>
  <ElementType name="major" dt:type="int" content="textOnly" />
  <ElementType name="minor" dt:type="int" content="textOnly" />
  <ElementType name="URLBase" dt:type="uri" content="textOnly" />
  <ElementType name="device" content="eltOnly">
    <element type="deviceType" />
    <element type="friendlyName" />
    <element type="manufacturer" />
    <element type="manufacturerURL" minOccurs="0" maxOccurs="1" />
    <element type="modelDescription" minOccurs="0" maxOccurs="1" />
    <element type="modelName" />
    <element type="modelNumber" minOccurs="0" maxOccurs="1" />
    <element type="modelURL" minOccurs="0" maxOccurs="1" />
    <element type="serialNumber" minOccurs="0" maxOccurs="1" />
    <element type="UDN" />
    <element type="UPC" minOccurs="0" maxOccurs="1" />
    <element type="iconList" minOccurs="0" maxOccurs="1" />
    <element type="serviceList" />
    <element type="deviceList" minOccurs="0" maxOccurs="1" />
    <element type="presentationURL" minOccurs="0" maxOccurs="1" />
  </ElementType>
  <ElementType name="deviceType" dt:type="uri" content="textOnly" />
  <ElementType name="friendlyName" dt:type="string" content="textOnly" />
  <ElementType name="manufacturer" dt:type="string" content="textOnly" />
  <ElementType name="manufacturerURL" dt:type="uri" content="textOnly" />
  <ElementType name="modelDescription" dt:type="string" content="textOnly" />
  <ElementType name="modelName" dt:type="string" content="textOnly" />
  <ElementType name="modelNumber" dt:type="string" content="textOnly" />
  <ElementType name="modelURL" dt:type="uri" content="textOnly" />
  <ElementType name="serialNumber" dt:type="string" content="textOnly" />
  <ElementType name="UDN" dt:type="uri" content="textOnly" />
  <ElementType name="UPC" dt:type="string" content="textOnly" />
  <ElementType name="iconList" content="eltOnly">
    <element type="icon" minOccurs="1" maxOccurs="*" />
  </ElementType>
  <ElementType name="icon" content="eltOnly">
    <element type="mimetype" />
    <element type="width" />
    <element type="height" />
    <element type="depth" />
    <element type="url" />
  </ElementType>
  <ElementType name="mimetype" dt:type="string" content="textOnly" />
  <ElementType name="width" dt:type="int" content="textOnly" />
  <ElementType name="height" dt:type="int" content="textOnly" />
  <ElementType name="depth" dt:type="int" content="textOnly" />
```

ROKU EXH. 1002

```
  <ElementType name="url" dt:type="uri" content="textOnly" />
  <ElementType name="serviceList" content="eltOnly">
    <element type="service" minOccurs="1" maxOccurs="*" />
  </ElementType>
  <ElementType name="service" content="eltOnly">
    <element type="serviceType" />
    <element type="serviceId" />
    <element type="SCPDURL" />
    <element type="controlURL" />
    <element type="eventSubURL" />
  </ElementType>
  <ElementType name="serviceType" dt:type="uri" content="textOnly" />
  <ElementType name="serviceId" dt:type="uri" content="textOnly" />
  <ElementType name="SCPDURL" dt:type="uri" content="textOnly" />
  <ElementType name="controlURL" dt:type="uri" content="textOnly" />
  <ElementType name="eventSubURL" dt:type="uri" content="textOnly" />
  <ElementType name="deviceList" content="eltOnly">
    <element type="device" minOccurs="1" maxOccurs="*" />
  </ElementType>
  <ElementType name="presentationURL" dt:type="uri" content="textOnly" />
</Schema>
```

ElementType
: Defines an element in the new, derived language. name attribute defines element name. dt:type attribute defines the data type for the value of element in the new, derived language.

element
: References an element for the purposes of declaring nesting. minOccurs attribute defines minimum number of times the element must occur; default is minOccurs = 1; optional elements have minOccurs = 0. maxOccurs attribute defines maximum number of times the element must occur; default is maxOccurs = 1; elements that can appear one or more times have maxOccurs = *.

2.7 Description: UPnP Template Language for services

The paragraphs above explain UPnP service descriptions and illustrate how one would be instantiated from a UPnP Service Template. Like UPnP Device Templates, UPnP Service Templates are produced by UPnP Forum working committees, and these templates are derived from the UPnP Template Language. This template language defines valid templates for devices and services. As explained above, the UPnP Template Language is written in XML syntax and is derived from XML Schema (Part 1: Structures, Part 2: Datatypes). Below is a listing of this language as it pertains to services. The elements it defines are used in UPnP Service Templates; they are colored green here, and they are colored green in the listing above. Below is where these elements are defined; above is where they are used.

Immediately following this is a brief explanation of the XML Schema elements, attributes, and values used. The reference to XML Schema at the end of the section has further details.

UPnP Template Language for services

```
<?xml version="1.0"?>
<Schema name="service-1-0"
    xmlns="urn:schemas-microsoft-com:xml-data"
    xmlns:dt="urn:schemas-microsoft-com:datatypes">
  <ElementType name="scpd" content="eltOnly">
    <element type="specVersion" />
    <element type="actionList" minOccurs="0" maxOccurs="1" />
    <element type="serviceStateTable" />
  </ElementType>
  <ElementType name="specVersion" content="eltOnly">
    <element type="major" />
    <element type="minor" />
```

ROKU EXH. 1002

```
  </ElementType>
  <ElementType name="major" dt:type="int" content="textOnly" />
  <ElementType name="minor" dt:type="int" content="textOnly" />
  <ElementType name="actionList" content="eltOnly">
    <element type="action" minOccurs="1" maxOccurs="*" />
  </ElementType>
  <ElementType name="action" content="eltOnly">
    <element type="name" />
    <element type="argumentList" minOccurs="0" maxOccurs="1" />
  </ElementType>
  <ElementType name="name" dt:type="string" content="textOnly" />
  <ElementType name="argumentList" content="eltOnly">
    <element type="argument" minOccurs="1" maxOccurs="*" />
  </ElementType>
  <ElementType name="argument" content="eltOnly">
    <element type="name" />
    <element type="direction" />
    <element type="retval" minOccurs="0" maxOccurs="1" />
    <element type="relatedStateVariable" />
  </ElementType>
  <ElementType name="direction" dt:type="string" content="textOnly" />
  <ElementType name="retval" content="empty" />
  <ElementType name="relatedStateVariable" dt:type="string" content="textOnly" />
  <ElementType name="serviceStateTable" content="eltOnly">
    <element type="stateVariable" minOccurs="1" maxOccurs="*" />
  </ElementType>
  <ElementType name="stateVariable" content="eltOnly">
    <element type="name" />
    <element type="dataType" />
    <element type="defaultValue" minOccurs="0" maxOccurs="1" />
    <group minOccurs="0" maxOccurs="1" order="one">
      <element type="allowedValueList" />
      <element type="allowedValueRange" />
    </group>
    <AttributeType name="sendEvents" />
    <attribute default="yes" type="sendEvents" required="no" />
  </ElementType>
  <ElementType name="dataType" dt:type="string" content="textOnly" />
  <ElementType name="defaultValue" dt:type="string" content="textOnly" />
  <ElementType name="allowedValueList" content="eltOnly">
    <element type="allowedValue" minOccurs="1" maxOccurs="*" />
  </ElementType>
  <ElementType name="allowedValue" content="textOnly" />
  <ElementType name="allowedValueRange" content="eltOnly">
    <element type="minimum" />
    <element type="maximum" />
    <element type="step" minOccurs="0" maxOccurs="1" />
  </ElementType>
  <ElementType name="minimum" dt:type="number" content="textOnly" />
  <ElementType name="maximum" dt:type="number" content="textOnly" />
  <ElementType name="step" dt:type="number" content="textOnly" />
</Schema>
```

attribute
: References an attribute in the new, derived language for the purposes of declaring in which elements it may appear. Like any XML element, the AttributeType element may have attributes of its own. Using the required attribute within this element indicates whether the attribute must be present; optional attributes have required = no.

AttributeType
: Defines an attribute in the new, derived language. Like any XML element, the AttributeType element may have attributes of its own. Using the name attribute within this element defines the name of the attribute as it will be used in the derived language.

element

ROKU EXH. 1002

References an element for the purposes of declaring nesting. minOccurs attribute defines minimum number of times the element must occur; default is minOccurs = 1; optional elements have minOccurs = 0. maxOccurs attribute defines maximum number of times the element must occur; default is maxOccurs = 1; elements that can appear one or more times have maxOccurs = *.

ElementType

Defines an element in the new, derived language. name attribute defines element name. dt:type attribute defines the data type for the value of element in the new, derived language. model attribute indicates whether elements in the new, derived language can contain elements not explicitly specified here; when only previously specific elements may be used, model = closed. content attribute indicates what content may contain; elements that contain only other elements have content = eltOnly; elements that contain only strings have content = textOnly.

group

Organizes content into a group to specify a sequence. minOccurs attribute defines minimum number of times the group must occur. maxOccurs attribute defines maximum number of times the group must occur. order attribute constrains the sequence of elements; when at most one element is allowed, order = one.

2.8 Description: Augmenting the UPnP Template Language

Some properties of services are difficult to capture in the XML Schema formalism. In particular, it is useful to describe the effect actions have on state variables. This procedural information is awkward to describe in a declarative language like XML, so below is a recommended vocabulary for UPnP Forum working committees to use when defining service actions or for UPnP vendors to use when they wish to document the effects of extra actions.

ASSIGN (*v*, *a*)

Variable *v* becomes the value of argument *a*, i.e., *v* = *a*. *v* and *a* must be the same data type.

DECREMENT (*v*)

Equivalent to INCREMENT (*v*) with allowedValueRange step treated as -step.

DECREMENT_BOUNDED (*v*)

Equivalent to INCREMENT_BOUNDED (*v*) with allowedValueRange step treated as -step.

DECREMENT_WRAP (*v*)

Equivalent to INCREMENT_WRAP (*v*) with allowedValueRange step treated as -step.

INCREMENT (*v*)

Variable *v* becomes the value of *v* plus allowedValueRange step, i.e., *v* = *v* + step. Equivalent to DECREMENT (*v*) with allowedValueRange step treated as -step. *v* must have a numeric data type and must have an allowedValueRange definition.

INCREMENT_BOUNDED (*v*)

Variable *v* becomes the value of *v* plus allowedValueRange step, i.e., *v* = *v* + step.
If step is greater than 0 and if *v* plus step would be greater than allowedValueRange maximum, then *v* becomes maximum.
If step is less than 0 and if *v* plus step would be less than allowedValueRange minimum, then *v* becomes minimum.
Equivalent to DECREMENT_BOUNDED (*v*) with allowedValueRange step treated as -step. *v* must have a numeric data type and must have an allowedValueRange definition.

INCREMENT_WRAP (*v*, *c*)

Variable *v* becomes the value of *v* plus allowedValueRange step, i.e., *v* = *v* + step.
If step is greater than 0, and if *v* plus step would be greater than allowedValueRange maximum, then *v* becomes minimum plus step minus 1, i.e., *v* = minimum + step - 1; if step is 1, this simplifies to *v* = minimum.
If step is less than 0 and if *v* plus step would be less than allowedValueRange minimum, then *v* becomes maximum plus step plus 1, i.e., *v* = maximum + step + 1; if step is -1, this simplifies to *v* = maximum.

Equivalent to DECREMENT_WRAP (*v*) with allowedValueRange step treated as -step. *v* must have a numeric data type and must have an allowedValueRange definition.

NEXT_STRING_BOUNDED (*v*)
: Variable *v* becomes the next allowedValue after the current value of *v*. If *v* was already the last allowedValue, then *v* does not change. *v* must be a string data type and must have an allowedValueList definition.

NEXT_STRING_WRAP (*v*)
: Variable *v* becomes the next allowedValue after the current value of *v*. If *v* was already the last allowedValue, then *v* becomes the first allowedValue. *v* must be a string data type and must have an allowedValueList definition.

PREV_STRING_BOUNDED (*v*)
: Variable *v* becomes the previous allowedValue before the current value of *v*. If *v* was already the first allowedValue, then *v* does not change. *v* must be a string data type and must have an allowedValueList definition.

PREV_STRING_WRAP (*v*)
: Variable *v* becomes the previous allowedValue before the current value of *v*. If *v* was already the first allowedValue, then *v* becomes the last allowedValue. *v* must be a string data type and must have an allowedValueList definition.

SET (*v*, *c*)
: Variable *v* becomes the value of constant *c*, i.e., *v* = *c*. *v* and *c* must be the same data type.

TOGGLE (*v*)
: Variable *v* becomes the boolean negation of the value of *v*, i.e., *v* = NOT *v*. *v* must be boolean.

2.9 Description: Retrieving a description

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

UPnP vendor [purple]
UPnP Forum [red]
UPnP Device Architecture [green]
HTTP [black]
TCP [black]
IP [black]

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

Using this protocol stack, retrieving the UPnP device description is simple: the control point issues an HTTP GET request to the URL in the discovery message, and the device returns its description in the body of an HTTP response. Similarly, to retrieve a UPnP service description, the control point issues an HTTP GET request to the URL in the device description, and the device returns the description in the body of an HTTP response. The headers and body for the response and request are explained in detail below.

ROKU EXH. 1002

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

```
GET path to description HTTP/1.1
HOST: host for description:port for description
ACCEPT-LANGUAGE: language preferred by control point
```

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

Listed below are details for the request line and headers appearing in the listing above. All header values are case sensitive except where noted.

Request line

GET
: Method defined by HTTP.

path to description
: Path component of device description URL (LOCATION header in discovery message) or of service description URL (SCPDURL element in device description). Single, relative URL.

HTTP/1.1
: HTTP version.

Headers

HOST
: Required. Domain name or IP address and optional port components of device description URL (LOCATION header in discovery message) or of service description URL (SCPDURL element of device description). If the port is empty or not given, port 80 is assumed.

ACCEPT-LANGUAGE
: Recommended for retrieving device descriptions. Preferred language(s) for description. If no description is available in this language, device may return a description in a default language. RFC 1766 language tag(s).

After a control point sends a request, the device takes the second step and responds with a copy of its description. Including expected transmission time, a device must respond within 30 seconds. If it fails to respond within this time, the control point should re-send the request. A device must send a response in the following format. Values in *italics* are placeholders for actual values.

```
HTTP/1.1 200 OK
CONTENT-LANGUAGE: language used in description
CONTENT-LENGTH: Bytes in body
CONTENT-TYPE: text/xml
DATE: when responded
```

The body of this response is a UPnP device or service description as explained in detail above.

Listed below are details for the headers appearing in the listing above. All header values are case sensitive except where noted.

Headers

CONTENT-LANGUAGE

ROKU EXH. 1002

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

CONTENT-LENGTH
: Required. Length of body in Bytes. Integer.

CONTENT-TYPE
: Required. Must be text/xml.

DATE
: Recommended. When response was generated. RFC 1123 date.

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

2.10 Description references

FXPP
: Flexible XML Processing Profile. Specifies that unknown XML elements and their sub elements must be ignored. IETF draft.

ISO 8601
: ISO (International Organization for Standardization). Representations of dates and times, 1988-06-15. Available at: <http://www.iso.ch/markete/8601.pdf>.

RFC 1123
: Includes format for dates, for, e.g., HTTP DATE header. IETF request for comments.

RFC 1766
: Format for language tag for, e.g., HTTP ACCEPT-LANGUAGE header. IETF request for comments.

RFC 2387
: Format for representing content type, e.g., mimetype element for an icon. IETF request for comments.

UPC
: Universal Product Code. 12-digit, all-numeric code that identifies the consumer package. Managed by the Uniform Code Council. <http://www.uc-council.org/main/ID_Numbers_and_Bar_Codes.html>.

XML
: Extensible Markup Language. W3C recommendation.

XML Schema (Part 1: Structures, Part 2: Datatypes)
: Grammar defining UPnP Template Language. Defined using XML. W3C working draft. Part 1: Structures. Part 2: Datatypes.

3. Control

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

Control is the third step in UPnP networking. Given knowledge of a device and its services, a control point can ask those services to invoke actions and the control point can poll those services for the values of their state variables. Invoking actions is a kind of remote procedure call; a control point sends the action to the device's service, and when the action has completed (or failed), the service returns any results or errors. Polling for the value of state variables is a special case of this scenario where the action and its results are predefined.

To control a device, a control point invokes an action on the device's service. To do this, a control point sends a suitable control message to the control URL for the service (provided in the controlURL sub element of service element of device description). In response, the service returns any results or errors from the action. The effects of the action, if any, may also be modeled by changes in the variables that describe the run-time state of the service. When these state variables change, events are published to all interested control points. This section explains the protocol stack for, and format of, control messages. The section on Eventing explains event publication.

To determine the current value of a state variable, a control point may poll the service. Similar to invoking an action, a control point sends a suitable query message to the control URL for the service. In response, the service provides the value of the variable; each service is responsible for keeping its state table consistent so control points can poll and receive meaningful values. This section also explains the format of these query messages. The section on eventing explains automatic notification of variable values.

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 a device cancels its advertisements, a control point must assume the device and its services are no longer available.

While UPnP does define a means to invoke actions and poll for values, UPnP does not specify or constrain the design of an API for applications running on control points; OS vendors may create APIs that suit their customer's needs.

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

3.1 Control: Protocols

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

UPnP vendor [purple]
UPnP Forum [red]

ROKU EXH. 1002

UPnP Device Architecture [green]
SOAP [blue]
HTTP [black]
TCP [black]
IP [black]

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

3.2 Control: Action

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

3.2.1 Control: Action: Invoke

The Simple Object Access Protocol (SOAP) defines the use of XML and HTTP for remote procedure calls. UPnP uses SOAP to deliver control messages to devices and return results or errors back to control points.

SOAP defines additional HTTP headers, and to ensure that these are not confused with other HTTP extensions, SOAP follows the HTTP Extension Framework and specifies a SOAP-unique URI in the MAN header and prefixes the HTTP method with M-. In this case, the method is M-POST. Using M-POST requires the HTTP server to find and understand the SOAP-unique URI and SOAP-specific headers.

To provide firewalls and proxies greater administrative flexibility, SOAP specifies that requests must first be attempted *without* the MAN header or M- prefix. If the request is rejected with a response of "405 Method Not Allowed", then a second request must be sent using the MAN header and M-prefix. If that request is rejected with a response of "501 Not Implemented" or "510 Not Extended", the request fails. (Other HTTP responses should be processed according to the HTTP specification.)

Below is a listing of a control message sent using the POST method (without the MAN header) followed by an explanation of the headers and body. This is immediately followed by a listing of a control message sent using the M-POST method and MAN header.

To invoke an action on a device's service, a control point must send a request with method POST in the following format. Values in *italics* are placeholders for actual values.

```
POST path of control URL HTTP/1.1
HOST: host of control URL:port of control URL
CONTENT-LENGTH: bytes in body
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>
```

ROKU EXH. 1002

```
      other in args and their values go here, if any
    </u:actionName>
  </s:Body>
</s:Envelope>
```

Listed below are details for the request line, headers, and body elements appearing in the listing above. All header values and element names are case sensitive; values are not case sensitive except where noted. Except where noted, the order of elements is insignificant. Except where noted, required elements must occur exactly once (no duplicates), and recommended or optional elements may occur at most once.

Request line

POST
: Method defined by HTTP.

path control URL
: Path component of URL for control for this service (controlURL sub element of service element of device description). Single, relative URL.

HTTP/1.1
: HTTP version.

Headers

HOST
: Required. Domain name or IP address and optional port components of URL for control for this service (controlURL sub element of service element of device description). If the port is empty or not given, port 80 is assumed.

ACCEPT-LANGUAGE
: (No ACCEPT-LANGUAGE header is used in control messages.)

CONTENT-LENGTH
: Required. Length of body in bytes. Integer.

CONTENT-TYPE
: Required. Must be text/xlm. Should include character coding used, e.g., utf-8.

MAN
: (No MAN header in request with method POST.)

SOAPACTION
: Required header defined by SOAP. Must be the service type, hash mark, and name of action to be invoked, all enclosed in double quotes. If used in a request with method M-POST, header name must be qualified with HTTP name space defined in MAN header. Single URI.

Body

Envelope
: Required element defined by SOAP. xmlns namespace attribute must be "http://schemas.xmlsoap.org/soap/envelope/". Must include encodingStyle attribute with value "http://schemas.xmlsoap.org/soap/encoding/". Contains the following sub elements:

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

        actionName
        : Required. Name of element is name of action to invoke. xmlns namespace attribute must be the service type enclosed in double quotes. Must be the first sub element of Body. Contains the following, ordered sub element(s):

ROKU EXH. 1002

argumentName
: Required if and only if action has in arguments. Value to be passed to action. Repeat once for each in argument. (Element name not qualified by a namespace; element nesting context is sufficient.) Single data type as defined by UPnP service description.

For future extensibility, when processing XML like the listing above, as specified by the Flexible XML Processing Profile (FXPP), devices and control points must ignore: (a) any unknown elements and their sub elements or content, and (b) any unknown attributes and their values.

If a request with POST is rejected with a response of "405 Method Not Allowed", then a control point must send a second request with method M-POST and MAN in the following format. Values in *italics* are placeholders for actual values.

```
M-POST path of control URL HTTP/1.1
HOST: host of control URL:port of control URL
CONTENT-LENGTH: bytes in body
CONTENT-TYPE: text/xml; charset="utf-8"
MAN: "http://schemas.xmlsoap.org/soap/envelope/"; ns=01
01-SOAPACTION: "urn:schemas-upnp-org:service:serviceType:v#actionName"
```

(Message body for request with method M-POST is the same as body for request with method POST. See above.)

Request line

M-POST
: Method defined by HTTP Extension Framework.

path of control URL
: Path component of URL for control for this service (controlURL sub element of service element of device description). Single, relative URL.

HTTP/1.1
: HTTP version.

Headers

HOST
: Required. Domain name or IP address and optional port components of URL for control for this service (controlURL sub element of service element of device description). If the port is empty or not given, port 80 is assumed.

ACCEPT-LANGUAGE
: (No ACCEPT-LANGUAGE header is used in control messages.)

CONTENT-LENGTH
: Required. Length of body in bytes. Integer.

CONTENT-TYPE
: Required. Must be text/xlm. Should include character coding used, e.g., utf-8.

MAN
: Required. Must be "http://schemas.xmlsoap.org/soap/envelope/". ns directive defines namespace (e.g., 01) for other SOAP headers (e.g., SOAPACTION).

SOAPACTION
: Required header defined by SOAP. Must be the service type, hash mark, and name of action to be invoked, all enclosed in double quotes. If used in a request with method M-POST, header name must be qualified with HTTP name space defined in MAN header. Single URI.

3.2.2 Control: Action: Response

ROKU EXH. 1002

The service must complete invoking the action and respond within 30 seconds, including expected transmission time. Actions that take longer than this should be defined to return early and send an event when complete. If the service fails to respond within this time, what the control point should do is application-specific. The service must send a response in the following format. Values in *italics* are placeholders for actual values.

```
HTTP/1.1 200 OK
CONTENT-LENGTH: bytes in body
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>out arg value</argumentName>
      other out args and their values go here, if any
    </u:actionNameResponse>
  </s:Body>
</s:Envelope>
```

Listed below are details for the response line, headers, and body elements appearing in the listing above. All header values and element names are case sensitive; values are not case sensitive except where noted. Except where noted, the order of elements is insignificant. Except where noted, required elements must occur exactly once (no duplicates), and recommended or optional elements may occur at most once.

Response line

HTTP/1.1
: HTTP version.

200 OK
: HTTP success code.

Headers

CONTENT-LANGUAGE
: (No CONTENT-LANGUAGE header is used in control messages.)

CONTENT-LENGTH
: Required. Length of body in bytes. Integer.

CONTENT-TYPE
: Required. Must be text/xlm. Should include character coding used, e.g., utf-8.

DATE
: Recommended. When response was generated. RFC 1123 date.

EXT
: Required. Confirms that the MAN header was understood. (Header only; no value.)

SERVER
: Required. Concatenation of OS name, OS version, UPnP/1.0, product name, and product version. String.

Body

Envelope
: Required element defined by SOAP. xmlns namespace attribute must be "http://schemas.xmlsoap.org/soap/envelope/". Must include encodingStyle attribute with value "http://schemas.xmlsoap.org/soap/encoding/". Contains the following sub elements:

ROKU EXH. 1002

Body

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

*actionName*Response

Required. Name of element is action name prepended to Response. xmlns namespace attribute must be service type enclosed in double quotes. Must be the first sub element of Body. Contains the following sub element:

argumentName

Required if and only if action has out arguments. Value returned from action. Repeat once for each out argument. If action has an argument marked as retval, this argument must be the first element. (Element name not qualified by a namespace; element nesting context is sufficient.) Single data type as defined by UPnP service description.

For future extensibility, when processing XML like the listing above, as specified by the Flexible XML Processing Profile (FXPP), devices and control points must ignore: (a) any unknown elements and their sub elements or content, and (b) any unknown attributes and their values.

If the service encounters an error while invoking the action sent by a control point, the service must send a response within 30 seconds, including expected transmission time. Out arguments must not be used to convey error information; out arguments must only be used to return data; error responses must be sent in the following format. Values in *italics* are placeholders for actual values.

```
HTTP/1.1 500 Internal Server Error
CONTENT-LENGTH: bytes in body
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>
```

Listed below are details for the response line, headers, and body elements appearing in the listing above. All header values and element names are case sensitive; values are not case sensitive except where noted. Except where noted, the order of elements is insignificant. Except where noted, required elements must occur exactly once (no duplicates), and recommended or optional elements may occur at most once.

Response line

HTTP/1.1

HTTP version.

500 Internal Server Error

ROKU EXH. 1002

HTTP error code.

Headers

CONTENT-LANGUAGE
: (No CONTENT-LANGUAGE header is used in control messages.)

CONTENT-LENGTH
: Required. Length of body in bytes. Integer.

CONTENT-TYPE
: Required. Must be text/xlm. Should include character coding used, e.g., utf-8.

DATE
: Recommended. When response was generated. RFC 1123 date.

EXT
: Required. Confirms that the MAN header was understood. (Header only; no value.)

SERVER
: Required. Concatenation of OS name, OS version, UPnP/1.0, product name, and product version. String.

Body

Envelope
: Required element defined by SOAP. xmlns namespace attribute must be "http://schemas.xmlsoap.org/soap/envelope/". Must include encodingStyle attribute with value "http://schemas.xmlsoap.org/soap/encoding/". Contains the following sub elements:

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

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

      faultcode
      : Required element defined by SOAP. Value must be qualified with the SOAP namespace. Must be Client.

      faultstring
      : Required element defined by SOAP. Must be UPnPError.

      detail
      : Required element defined by SOAP.

        UPnPError
        : Required element defined by UPnP.

          errorCode
          : Required element defined by UPnP. Code identifying what error was encountered. See table immediately below for values. Integer.

          errorDescription
          : Recommended element defined by UPnP. Short description. See table immediately below for values. String. Recommend < 256 characters.

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

errorCode	errorDescription	Description

ROKU EXH. 1002

401	Invalid Action	No action by that name at this service.
402	Invalid Args	Could be any of the following: not enough in args, too many in args, no in arg by that name, one or more in args are of the wrong data type.
403	Out of Sync	Out of synchronization.
501	Action Failed	May be returned in current state of service prevents invoking that action.
600-699	*TBD*	Common action errors. Defined by UPnP Forum Technical Committee.
700-799	*TBD*	Action-specific errors for standard actions. Defined by UPnP Forum working committee.
800-899	*TBD*	Action-specific errors for non-standard actions. Defined by UPnP vendor.

For future extensibility, when processing XML like the listing above, as specified by the Flexible XML Processing Profile (FXPP), devices and control points must ignore: (a) any unknown elements and their sub elements or content, and (b) any unknown attributes and their values.

3.3 Control: Query for variable

In addition to invoking actions on a device's service, control points may also poll the service for the value of a state variable by sending a query message. A query message may query only one state variable; multiple query messages must be sent to query multiple state variables.

This query message is decoupled from the service's eventing (if any). If a variable is moderated, then querying for the value of the variable will generally yield more up-to-date values than those received via eventing. The section on Eventing describes event moderation.

3.3.1 Control: Query: Invoke

To query for the value of a state variable, a control point must send a request in the following format. Values in *italics* are placeholders for actual values.

```
POST path of control URL HTTP/1.1
HOST: host of control URL:port of control URL
CONTENT-LENGTH: bytes in body
CONTENT-TYPE: text/xml; charset="utf-8"
SOAPACTION: "urn:schemas-upnp-org:control-1-0#QueryStateVariable"

<s:Envelope
    xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"
    s:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <s:Body>
    <u:QueryStateVariable xmlns:u="urn:schemas-upnp-org:control-1-0">
      <u:varName>variableName</u:varName>
    </u:QueryStateVariable>
  </s:Body>
</s:Envelope>
```

Listed below are details for the request line, headers, and body elements appearing in the listing above. All header values and element names are case sensitive; values are not case sensitive except where noted. Except where noted, the order of elements is insignificant. Except where noted, required elements must occur exactly once (no duplicates), and recommended or optional elements may occur at most once.

Request line

POST

ROKU EXH. 1002

Method defined by HTTP.

path of control URL

Path component of URL for control for this service (controlURL sub element of service element of device description). Single, relative URL.

HTTP/1.1

HTTP version.

Headers

HOST

Required. Domain name or IP address and optional port components of URL for control for this service (controlURL sub element of service element of device description). If the port is empty or not given, port 80 is assumed.

ACCEPT-LANGUAGE

(No ACCEPT-LANGUAGE header is used in control messages.)

CONTENT-LENGTH

Required. Length of body in bytes. Integer.

CONTENT-TYPE

Required. Must be text/xlm. Should include character coding used, e.g., utf-8.

MAN

(No MAN header in request with method POST.)

SOAPACTION

Required header defined by SOAP. Must be "urn:schemas-upnp-org:control-1-0#QueryStateVariable". If used in a request with method M-POST, header name must be qualified with HTTP name space defined in MAN header. Single URI.

Body

Envelope

Required element defined by SOAP. xmlns namespace attribute must be "http://schemas.xmlsoap.org/soap/envelope/". Must include encodingStyle attribute with value "http://schemas.xmlsoap.org/soap/encoding/". Contains the following sub elements:

Body

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

QueryStateVariable

Required element defined by UPnP. Action name. xmlns namespace attribute must be "urn:schemas-upnp-org:control-1-0". Must be the first sub element of Body. Contains the following, ordered sub element:

varName

Required element defined by UPnP. Variable name. Must be qualified by QueryStateVariable namespace. Values is name of state variable to be queried. String.

For future extensibility, when processing XML like the listing above, as specified by the Flexible XML Processing Profile (FXPP), devices and control points must ignore: (a) any unknown elements and their sub elements or content, and (b) any unknown attributes and their values.

If a request with POST is rejected with a response of "405 Method Not Allowed", then a control point must send a second request with method M-POST and MAN as explained above.

3.3.2 Control: Query: Response

ROKU EXH. 1002

To answer a query for the value of a state variable, the service must respond within 30 seconds, including expected transmission time. If the service fails to respond within this time, what the control point should do is application-specific. The service must send a response in the following format. Values in *italics* are placeholders for actual values.

```
HTTP/1.1 200 OK
CONTENT-LENGTH: bytes in body
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:QueryStateVariableResponse xmlns:u="urn:schemas-upnp-org:control-1-0">
      <return>variable value</return>
    </u:QueryStateVariableResponse>
  </s:Body>
</s:Envelope>
```

Listed below are details for the response line, headers, and body elements appearing in the listing above. All header values and element names are case sensitive; values are not case sensitive except where noted. Except where noted, the order of elements is insignificant. Except where noted, required elements must occur exactly once (no duplicates), and recommended or optional elements may occur at most once.

Response line

HTTP/1.1
    HTTP version.
200 OK
    HTTP success code.

Headers

CONTENT-LANGUAGE
    (No CONTENT-LANGUAGE header is used in control messages.)
CONTENT-LENGTH
    Required. Length of body in bytes. Integer.
CONTENT-TYPE
    Required. Must be text/xlm. Should include character coding used, e.g., utf-8.
DATE
    Recommended. When response was generated. RFC 1123 date.
EXT
    Required. Confirms that the MAN header was understood. (Header only; no value.)
SERVER
    Required. Concatenation of OS name, OS version, UPnP/1.0, product name, and product version. String.

Body

Envelope
    Required element defined by SOAP. xmlns namespace attribute must be "http://schemas.xmlsoap.org/soap/envelope/". Must include encodingStyle attribute with value "http://schemas.xmlsoap.org/soap/encoding/". Contains the following sub elements:

ROKU EXH. 1002

Body

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

QueryStateVariableResponse

Required element defined by UPnP and SOAP. xmlns namespace attribute must be "urn:schemas-upnp-org:control-1-0". Must be the first sub element of Body. Contains the following sub element:

return

Required element defined by UPnP. (Element name not qualified by a namespace; element nesting context is sufficient.) Value is current value of the state variable specified in varName element in request.

For future extensibility, when processing XML like the listing above, as specified by the Flexible XML Processing Profile (FXPP), devices and control points must ignore: (a) any unknown elements and their sub elements or content, and (b) any unknown attributes and their values.

If the service cannot provide a value for the variable, then the service must send a response within 30 seconds, including expected transmission time. The response must be sent in the following format. Values in *italics* are placeholders for actual values.

```
HTTP/1.1 500 Internal Server Error
CONTENT-LENGTH: bytes in body
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>
```

Listed below are details for the response line, headers, and body elements appearing in the listing above. All header values and element names are case sensitive; values are not case sensitive except where noted. Except where noted, the order of elements is insignificant. Except where noted, required elements must occur exactly once (no duplicates), and recommended or optional elements may occur at most once.

Response line

HTTP/1.1

HTTP version.

500 Internal Server Error

HTTP error code.

ROKU EXH. 1002

Headers

CONTENT-LANGUAGE
: (No CONTENT-LANGUAGE header is used in control messages.)

CONTENT-LENGTH
: Required. Length of body in bytes. Integer.

CONTENT-TYPE
: Required. Must be text/xlm. Should include character coding used, e.g., utf-8.

DATE
: Recommended. When response was generated. RFC 1123 date.

EXT
: Required. Confirms that the MAN header was understood. (Header only; no value.)

SERVER
: Required. Concatenation of OS name, OS version, UPnP/1.0, product name, and product version. String.

Body

Envelope
: Required element defined by SOAP. xmlns namespace attribute must be "http://schemas.xmlsoap.org/soap/envelope/". Must include encodingStyle attribute with value "http://schemas.xmlsoap.org/soap/encoding/". Contains the following sub elements:

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

    Fault
    : Required element defined by SOAP. Why the service did not return a value for the variable. Should be qualified with SOAP namespace. Contains the following sub elements:

      faultcode
      : Required element defined by SOAP. Value should be qualified with SOAP namespace. Must be Client.

      faultstring
      : Required element defined by SOAP. Generic UPnP string describing errorCode. See table immediately below for values.

      detail
      : Required element defined by SOAP. Contains the following sub elements:

        UPnPError
        : Required element defined by UPnP. Contains the following sub elements:

          errorCode
          : Required element defined by UPnP. Code identifying what error was encountered. See table immediately below for values. Integer.

          errorDescription
          : Recommended element defined by UPnP. Short description. See table immediately below for values. String. Recommend < 256 characters.

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

errorCode	errorDescription	Description
404	Invalid Var	No state variable by that name at this service.

ROKU EXH. 1002

600-624	*TBD*	Common action errors. Defined by UPnP Forum Technical Committee.
625-649	*TBD*	Reserved for future use.
650-674	*TBD*	Action-specific errors for standard actions. Defined by UPnP Forum working committee.
675-699	*TBD*	Action-specific errors for non-standard actions. Defined by UPnP vendor.

For future extensibility, when processing XML like the listing above, as specified by the Flexible XML Processing Profile (FXPP), devices and control points must ignore: (a) any unknown elements and their sub elements or content, and (b) any unknown attributes and their values.

3.4 Control references

FXPP
: Flexible XML Processing Profile. Specifies that unknown XML elements and their sub elements must be ignored. IETF draft.

HTTP Extension Framework
: Describes a generic extension mechanism for HTTP. W3C request for comments.

RFC 1123
: Includes format for dates, for, e.g., HTTP DATE header. IETF request for comments.

SOAP
: Simple Object Access Protocol. Defines a protocol in XML, over HTTP, for remote procedure calls. IETF draft and W3C Technical Report.

XML
: Extensible Markup Language. W3C recommendation.

4. Eventing

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

After a control point has (1) discovered a device and (2) retrieved a description of the device and its services, the control point has the essentials for eventing. As the section on Description explains, a UPnP service description includes a list of actions the service responds to and a list of variables that model the state of the service at run time. If one or more of these state variables are evented, then the service publishes updates when these variables change, and a control point may subscribe to receive this information. Throughout this section, *publisher* refers to the source of the events (typically a device's service), and *subscriber* refers to the destination of events (typically a control point).

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

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

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

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

UPnP vendor [purple]
UPnP Forum [red]

UPnP Device Architecture [green]
HTTP [black] GENA [navy]
TCP [black]
IP [black]

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

The remainder of this section first explains subscription, including details of subscription messages, renewal messages, and cancellation messages. Second, it explains in detail how event messages are formatted and sent to control points, and the initial event message. Finally, it explains the UPnP Template Language as it pertains to eventing.

4.1 Eventing: Subscription

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

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

unique subscription identifier
: Required. Must be unique over the lifetime of the subscription, however long or short that may be. Generated by publisher in response to subscription message. Recommend universally-unique identifiers to ensure uniqueness. Single URI.

delivery URL for event messages
: Required. Provided by subscriber in subscription message. Single URL.

event key
: Required. Key is 0 for initial event message. Key must be sequentially numbered for each subsequent event message; subscribers can verify that no event messages have been lost if the subscriber has received sequentially numbered event keys. Must wrap to 1. Should be 4 Bytes (32 bits). Single integer.

subscription duration
: Required. Amount of time, or duration until subscription expires. Single integer or keyword infinite.

The publisher should accept as many subscriptions as it can reasonably maintain and deliver.

The publisher may wish to persist subscriptions across power failures. While control points can recover from complete network failure, if the problem is brief and localized to the device, reusing stored subscriptions may speed recovery.

The list of subscribers is updated via subscription, renewal, and cancellation messages explained immediately below and event messages explained later in this section.

To subscribe to eventing for a service, a subscriber sends a *subscription message* containing a URL for the publisher, a service identifier for the publisher, and a delivery URL for event messages. The subscription message may also include a requested duration for the subscription. The URL and service identifier for the publisher come from a description message. As the section on Description explains, a description message contains a device description. A device description contains (among other things), for each service, an eventing URL (in the eventSubURL element) and a service identifier (in the serviceId element); these correspond to the URL and service identifier for the publisher, respectively. The URL for the publisher must be unique to a particular service within this device.

The subscription message is a request to receive all event messages. No mechanism is provided to subscribe to event messages on a variable-by-variable basis. A subscriber is sent all event messages from the service. This is one factor to be considered when designing a service.

If the subscription is accepted, the publisher responds with unique identifier for this subscription and a duration for this subscription. A duration should be chosen that matches assumptions about how frequently control points are removed from the network; if control points are removed every few minutes, then the duration should be similarly short, allowing a publisher to rapidly deprecate any expired subscribers; if control points are expected to be semi-permanent, then the duration should be very long, minimizing the processing and traffic associated with renewing subscriptions.

As soon as possible after the subscription is accepted, the publisher also sends the first, or *initial* event message to the subscriber. This message includes the names and current values for all evented variables. (The data type and range for each variable is described in a service description. The section on Description explains this in more detail.)

To keep the subscription active, a subscriber must renew its subscription before the subscription expires by sending a renewal message. The renewal message is send to the same URL as the subscription message, but the renewal message does not include a delivery URL for event messages; instead the renewal message includes the subscription identifier. The response for a renewal message is the same as one for a subscription message.

If a subscription expires, the subscription identifier becomes invalid, and the publisher stops sending event messages to the subscriber and can clean up its list of subscribers. If the subscriber tries to send any message other than a subscription message, the publisher will reject the message because the subscription identifier is invalid.

When a subscriber no longer needs eventing from a particular service, the subscriber should cancel its subscription. Canceling a subscription generally reduces service, control point, and network load. If a subscriber is removed abruptly from the network, it might be impossible to send a cancellation message. As a fallback, the subscription will eventually expire on its own unless renewed.

Subscribers should monitor discovery messages from the publisher. If the publisher cancels its advertisements, subscribers should assume that their subscriptions have been effectively cancelled.

Below is an explanation of the specific format of requests, responses, and errors for subscription, renewal, and cancellation messages.

4.1.1 Eventing: Subscribing: SUBSCRIBE with NT and CALLBACK

For each service in a device, a description message contains an eventing URL (eventSubURL sub element of service element in the device description) and the UPnP service identifier (serviceId sub element in service element in device description). To subscribe to eventing for a particular service, a subscription message is sent to that service's eventing URL. (Note that the eventing URL may be relative to the base URL.) The message contains that service's identifier as well as a delivery URL for event messages. A subscription message may also include a requested subscription duration.

To subscribe to eventing for a service, a subscriber must send a request with method SUBSCRIBE and NT and CALLBACK headers in the following format. Values in *italics* are placeholders for actual values.

```
SUBSCRIBE publisher path HTTP/1.1
HOST: publisher host:publisher port
CALLBACK: <delivery URL>
NT: upnp:event
TIMEOUT: Second-requested subscription duration
```

(No body for request with method SUBSCRIBE, but note that the message must have a blank line following the last HTTP header.)

Listed below are details for the request line and headers appearing in the listing above. All header values are case sensitive except where noted.

Request line

SUBSCRIBE
: Method defined by GENA. Initiate or renew a subscription.

publisher path
: Path component of eventing URL (eventSubURL sub element in service element in device description). Single, relative URL.

HTTP/1.1
: HTTP version.

Headers

HOST
: Required. Domain name or IP address and optional port components of eventing URL (eventSubURL sub element in service element in device description). If the port is missing or empty, port 80 is assumed.

CALLBACK
: Required header defined by GENA. Location to send event messages to. Defined by UPnP vendor. If there is more than 1 URL, when the service sends events, it will try these URLs in order until one succeeds. One or more URLs separated by angle brackets.

NT
: Required header defined by GENA. Notification Type. Must be upnp:event.

SID
: (No SID header is used to subscribe.)

TIMEOUT
: Recommended. Requested duration until subscription expires, either number of seconds or infinite. Recommendation by a UPnP Forum working committee. Defined by UPnP vendor. Keyword Second-followed by an integer (no space) or keyword infinite.

If there are enough resources to maintain the subscription, the publisher should accept it. To accept the subscription, the publisher assigns a unique identifier for the subscription, assigns a duration for the subscription, and sends an initial event message (explained in detail later in this section). To accept a subscription request, a publisher must send a response in the following format within 30 seconds, including expected transmission time. Values in *italics* are placeholders for actual values.

```
HTTP/1.1 200 OK
DATE: when response was generated
SERVER: OS/version UPnP/1.0 product/version
SID: uuid:subscription-UUID
TIMEOUT: Second-actual subscription duration
```

(No body for response to a request with method SUBSCRIBE, but note that the message must have a blank line following the last HTTP header.)

ROKU EXH. 1002

Listed below are details for headers appearing in the listing above. All header values are case sensitive except where noted.

Headers

DATE
: Recommended. When response was generated. RFC 1123 date.

SERVER
: Required. Concatenation of OS name, OS version, UPnP/1.0, product name, and product version. String.

SID
: Required header defined by GENA. Subscription identifier. Must be universally unique. Must begin with uuid:. Defined by UPnP vendor. Single URI.

TIMEOUT
: Required. Actual duration until subscription expires, either number of seconds or infinite. Recommendation by a UPnP Forum working committee. Defined by UPnP vendor. Should be > 1800 seconds (30 minutes). Keyword Second- followed by an integer (no space) or keyword infinite.

If a publisher cannot accept another subscriber, or if there is an error with the subscription request, the publisher must send a response with one of the following errors. The response must be sent within 30 seconds, including expected transmission time.

Errors

Incompatible headers
: 400 Bad Request. If SID header and one of NT or CALLBACK headers are present, the publisher must respond with HTTP error 400 Bad Request.

Missing or invalid CALLBACK
: 412 Precondition Failed. If CALLBACK header is missing or does not contain a valid HTTP URL, the publisher must respond with HTTP error 412 Precondition Failed.

Invalid NT
: 412 Precondition Failed. If NT header does not equal upnp:event, the publisher must respond with HTTP error 412 Precondition Failed.

Unable to accept subscription
: 5xx. If a publisher is not able to accept a subscription, it must respond with a HTTP 500-series error code.

Other errors may be returned by layers in the protocol stack below UPnP. Consult documentation on those protocols for details.

4.1.2 Eventing: Renewing a subscription: SUBSCRIBE with SID

To renew a subscription to eventing for a particular service, a renewal message is sent to that service's eventing URL. (Note that the eventing URL may be relative to the base URL.) However, unlike an initial subscription message, a renewal message does not contain either the service's identifier nor a delivery URL for event messages. Instead, the message contains the *subscription* identifier assigned by the publisher, providing an unambiguous reference to the subscription to be renewed. Like a subscription message, a renewal message may also include a requested subscription duration.

The renewal message uses the same method as the subscription message, but the two messages use a disjoint set of headers; renewal uses SID and subscription uses NT and CALLBACK. A message that includes SID and either of NT or CALLBACK headers is an error.

To renew a subscription to eventing for a service, a subscriber must send a request with method SUBSCRIBE and SID header in the following format. Values in *italics* are placeholders for actual values.

```
SUBSCRIBE publisher path HTTP/1.1
HOST: publisher host:publisher port
SID: uuid:subscription UUID
TIMEOUT: Second-requested subscription duration
```

(No body for method with request SUBSCRIBE, but note that the message must have a blank line following the last HTTP header.)

Listed below are details for the request line and headers appearing in the listing above. All header values are case sensitive except where noted.

Request line

SUBSCRIBE
: Method defined by GENA. Initiate or renew a subscription.

publisher path
: Path component of eventing URL (eventSubURL sub element in service element in device description). Single, relative URL.

HTTP/1.1
: HTTP version.

Headers

HOST
: Required. Domain name or IP address and optional port components of eventing URL (eventSubURL sub element in service element in device description). If the port is missing or empty, port 80 is assumed.

CALLBACK
: (No CALLBACK header is used to renew an event subscription.)

NT
: (No NT header is used to renew an event subscription.)

SID
: Required header defined by GENA. Subscription identifier. Must be the subscription identifier assigned by publisher in response to subscription request. Must be universally unique. Must begin with uuid:. Defined by UPnP vendor. Single URI.

TIMEOUT
: Recommended. Requested duration until subscription expires, either number of seconds or infinite. Recommendation by a UPnP Forum working committee. Defined by UPnP vendor. Keyword Second- followed by an integer (no space) or keyword infinite.

To accept a renewal, the publisher reassigns a duration for the subscription and must send a response in the same format as a response to a request for a new subscription. (No initial event message.)

If a publisher cannot accept the renewal, or if there is an error with the renewal request, the publisher must send a response with one of the following errors. The response must be sent within 30 seconds, including expected transmission time.

Errors

Incompatible headers
: 400 Bad Request. If SID header and one of NT or CALLBACK headers are present, the publisher must respond with HTTP error 400 Bad Request.

Invalid SID

412 Precondition Failed. If a SID does not correspond to a known, un-expired subscription, the publisher must respond with HTTP error 412 Precondition Failed.

Missing SID
: 412 Precondition Failed. If the SID header is missing or empty, the publisher must respond with HTTP error 412 Precondition Failed.

Unable to accept renewal
: 5xx. If the publisher is not able to accept a renewal, it must respond with a HTTP 500-series error code.

Other errors may be returned by layers in the protocol stack below UPnP. Consult documentation on those protocols for details.

4.1.3 Eventing: Canceling a subscription: UNSUBSCRIBE

When eventing is no longer needed from a particular service, a cancellation message should be sent to that service's eventing URL. (Note that the eventing URL may be relative to the base URL.) The message contains the subscription identifier. Canceling a subscription generally reduces service, control point, and network load. If a control point is removed abruptly from the network, it might be impossible to send a cancellation message. As a fallback, the subscription will eventually expire on its own unless renewed.

To cancel a subscription to eventing for a service, a subscriber should send a request with method UNSUBSCRIBE in the following format. Values in *italics* are placeholders for actual values.

```
UNSUBSCRIBE publisher path HTTP/1.1
HOST: publisher host:publisher port
SID: uuid:subscription UUID
```

(No body for request with method UNSUBSCRIBE, but note that the message must have a blank line following the last HTTP header.)

Listed below are details for the request line and headers appearing in the listing above. All header values are case sensitive except where noted.

Request line

UNSUBSCRIBE
: Method defined by GENA. Cancel a subscription.

publisher path
: Path component of eventing URL (eventSubURL sub element in service element in device description). Single, relative URL.

HTTP/1.1
: HTTP version.

Headers

HOST
: Required. Domain name or IP address and optional port components of eventing URL (eventSubURL sub element in service element in device description). If the port is missing or empty, port 80 is assumed.

CALLBACK
: (No CALLBACK header is used to cancel an event subscription.)

NT
: (No NT header is used to cancel an event subscription.)

SID

ROKU EXH. 1002

Required header defined by GENA. Subscription identifier. Must be the subscription identifier assigned by publisher in response to subscription request. Must be universally unique. Must begin with uuid:. Defined by UPnP vendor. Single URI.

TIMEOUT

(No TIMEOUT header is used to cancel an event subscription.)

To cancel a subscription, a publisher must send a response in the following format within 30 seconds, including expected transmission time.

```
HTTP/1.1 200 OK
```

If there is an error with the cancellation request, the publisher must send a response with one of the following errors. The response must be sent within 30 seconds, including expected transmission time.

Errors

Incompatible headers

400 Bad Request. If SID header and one of NT or CALLBACK headers are present, the publisher must respond with HTTP error 400 Bad Request.

Invalid SID

412 Precondition Failed. If a SID does not correspond to a known, un-expired subscription, the publisher must respond with HTTP error 412 Precondition Failed.

Missing SID

412 Precondition Failed. If the SID header is missing or empty, the publisher must respond with HTTP error 412 Precondition Failed.

Other errors may be returned by layers in the protocol stack below UPnP. Consult documentation on those protocols for details.

4.2 Eventing: Event messages

A service publishes changes to its state variables by sending event messages. These messages contain the names of one or more state variables and the current value of those variables. Event messages should be sent as soon as possible to get accurate information about the service to subscribers and allow subscribers to display a responsive user interface. If the value of more than one variable is changing at the same time, the publisher should bundle these changes into a single event message to reduce processing and network traffic.

As explained above, an initial event message is sent when a subscriber first subscribes; this event message contains the names and values for all evented variables and allows the subscriber to initialize its model of the state of the service. This message should be sent as soon as possible after the publisher accepts a subscription.

Event messages are tagged with an event key. A separate event key must be maintained by the publisher for each subscription to facilitate error detection (as explained below). The event key for a subscription is initialized to 0 when the publisher sends the initial event message. For each subsequent event message, the publisher increments the event key for a subscription, and includes that updated key in the event message. Any implementation of event keys should handle overflow and wrap the event key back to 1 (not 0). Subscribers must also handle this special case when the next event key is not an increment of the previous key. Should be implemented as a 4 Byte (32 bit) integer.

If there is no response from a subscriber to the event message, the publisher should continue to send event messages to the subscriber until the subscription expires.

To repair an event subscription, e.g., if a subscriber has missed one or more event messages, a subscriber must unsubscribe and re-subscribe. By doing so, the subscriber will get a new subscription identifier, a new initial event message, and a new event key.

4.2.1 Eventing: Event messages: NOTIFY

To send an event message, a publisher must send a request with method NOTIFY in the following format. Values in *italics* below are placeholders for actual values.

```
NOTIFY delivery path HTTP/1.1
HOST: delivery host:delivery port
CONTENT-TYPE: text/xml
CONTENT-LENGTH: Bytes in body
NT: upnp:event
NTS: upnp:propchange
SID: uuid:subscription-UUID
SEQ: event key

<e:propertyset xmlns:e="urn:schemas-upnp-org:event-1-0">
  <e:property>
    <variableName>new value</variableName>
  </e:property>
  Other variable names and values (if any) go here.
</e:propertyset>
```

Listed below are details for the request line, headers, and body elements appearing in the listing above. All header values are case sensitive except where noted. All body elements and attributes are case sensitive; body values are not case sensitive except where noted. Except where noted, the order of elements is insignificant. Except where noted, required elements must occur exactly once (no duplicates), and recommended or optional elements may occur at most once.

Request line

NOTIFY
: Method defined by GENA. Notify client about event.

delivery path
: Path component of delivery URL (CALLBACK header in subscription message). Destination for event message. Single, relative URL.

HTTP/1.1
: HTTP version.

Headers

HOST
: Required. Domain name or IP address and optional port components of delivery URL (CALLBACK header in subscription message). If the port is missing or empty, port 80 is assumed.

ACCEPT-LANGUAGE
: (No ACCEPT-LANGUAGE header is used in event messages.)

CONTENT-LENGTH
: Required. Length of body in Bytes. Integer.

CONTENT-TYPE
: Required. Must be text/xml.

NT
: Required header defined by GENA. Notification Type. Must be upnp:event.

NTS
: Required header defined by GENA. Notification Sub Type. Must be upnp:propchange.

ROKU EXH. 1002

SID
: Required header defined by GENA. Subscription identifier. Must be universally unique. Must begin with uuid:. Defined by UPnP vendor. Single URI.

SEQ
: Required header defined by UPnP. Event key. Must be 0 for initial event message. Must be incremented by 1 for each event message sent to a particular subscriber. Should be 8 Bytes long. To prevent overflow, must be wrapped to 1. Single integer.

Body

propertyset
: Required. xmlns namespace attribute must be urn:schemas-upnp-org:event-1-0. All sub elements must be qualified with this namespace. Contains the following sub element.

  property
  : Required. Repeat once for each variable name and value in the event message. Must be qualified by propertyset namespace. Contains the following sub element.

    variableName
    : Required. Element is name of a state variable that changed (name sub element of stateVariable element in service description). Must be qualified by propertyset namespace. Values is the new value for this state variable. Single data type as specified by UPnP service description.

For future extensibility, when processing XML like the listing above, as specified by the Flexible XML Processing Profile (FXPP), devices and control points must ignore: (a) any unknown elements and their sub elements or content, and (b) any unknown attributes and their values.

To acknowledge receipt of this event message, a subscriber must respond within 30 seconds, including expected transmission time. If a subscriber does not respond within 30 seconds, the publisher should abandon sending this message to the subscriber but should keep the subscription active and send future event messages the subscriber until the subscription expires or is cancelled. The subscriber must send a response in the following format.

```
HTTP/1.1 200 OK
```

(No body for a request with method NOTIFY, but note that the message must have a blank line following the last HTTP header.)

If there is an error with the event message, the subscriber must respond with one of the following errors. The response must be sent within 30 seconds, including expected transmission time.

Errors

Missing SID
: 412 Precondition Failed. If the SID header is missing or empty, the subscriber must respond with HTTP error 412 Precondition Failed.

Invalid SID
: 412 Precondition Failed. If a SID does not correspond to a known subscription, the subscriber must respond with HTTP error 412 Precondition Failed. (Service must terminate this SID when it receives this error response.)

Missing NT or NTS header
: 400 Bad Request. If the NT or NTS header is missing, the subscriber must respond with HTTP error 400 Bad Request.

Invalid NT header
: 412 Precondition Failed. If NT header does not equal upnp:event, the subscriber must respond with HTTP error 412 Precondition Failed.

Invalid NTS header
: 412 Precondition Failed. If NTS header does not equal upnp:propchange, the subscriber must respond with HTTP error 412 Precondition Failed.

Other errors may be returned by layers in the protocol stack below UPnP. Consult documentation on those protocols for details.

4.3 Eventing: UPnP Template Language for eventing

The UPnP Template Language defines well-formed templates for devices and services. To a lesser extent, it also provides a template for the body of event messages. The section on Description explains the UPnP Template Language as it pertains to devices and services. As explained in that section, the UPnP Template Language is written in XML syntax and is derived from XML Schema (Part 1: Structures, Part 2: Datatypes). Below is a listing of this language as it pertains to eventing. The elements it defines are used in event messages; they are colored green here, and they are colored green in the listing above. Below is where these elements are defined (though it is a minimal definition); above is where they are used.

Immediately following this is a brief explanation of the XML Schema elements, attributes, and values used. The reference to XML Schema at the end of this section has further details.

UPnP Template Language for eventing

```
<?xml version="1.0" ?>
<Schema name="urn:schemas-upnp-org:event-1-0"
    xmlns="urn:schemas-microsoft-com:xml-data"
    xmlns:dt="urn:schemas-microsoft-com:datatypes">
  <ElementType name="propertyset" content="eltOnly">
    <element type="property" minOccurs="1" maxOccurs="*" />
  </ElementType>
  <ElementType name="property" content="eltOnly" />
</Schema>
```

element
: References an element for the purposes of declaring nesting. maxOccurs attribute defines maximum number of times the element must occur; default is maxOccurs = 1; elements that can appear one or more times have maxOccurs = *.

ElementType
: Defines an element in the new, derived language. name attribute defines element name. model attribute indicates whether elements in the new, derived language can contain elements not explicitly specified here; when only unspecified sub elements may be included, model=open. content attribute indicates what content may contain; elements that contain only other elements have content = eltOnly.

As explained in the section on Description, the UPnP Template Language for services also specifies a sendEvents attribute for a state variable. The default value for this attribute is yes. To denote that a state variable is evented, the value of this attribute is yes (or the attribute is omitted) in a service description; to denote that a state variable is non-evented, the value is no. Note that if all of a service's state variables are non-evented, the service has nothing to publish, and control points cannot subscribe and will not receive event messages from the service.

4.4 Eventing: Augmenting the UPnP Template Language

ROKU EXH. 1002

It is useful to augment the description of devices and services with annotations that are not captured in the UPnP Template Language. To a lesser extent, there is value in these annotations to capture event filtering, or moderation.

As explained above, some state variables may change value too rapidly for eventing to be useful. Below is a recommended vocabulary for UPnP Forum working committees or UPnP vendors to document moderation in the number of event messages sent due to changes in a variables value.

maximumRate = *n*
: Optional. State variable *v* will not be part of an event message more often than *n* seconds. If *v* is the only variable changing, then an event message will not be generated more often than every *n* seconds. If *v* ceases to change after an event message has been sent but before *n* seconds have transpired, an event message must be sent with the new value of *v*. Recommended for variables that model continuously changing properties. Single integer.

minimumDelta = *n*
: Optional. State variable *v* will not be part of an event message unless its value has changed by more than *n* * allowedValueRange step since the last time an event message was sent that included *v*, e.g., unless *v* has been incremented *n* times. (cf. INCREMENT, INCREMENT_BOUNDED, and INCREMENT_WRAP explained in the section on Control.) Only defined variables with number and real data type. Recommended for variables that model counters. Single integer.

The publisher can send out any changed moderated variable when an event goes out. The publisher should make its best attempt to meet moderation rules described above, but the publisher can flush recent changes when it sends out events.

Note that moderation affects events only and not state table updates. Specifically, QueryStateVariable may return a more current value than published via eventing. Put another way, moderation means that not all state table changes result in events.

Decisions about which variables to event and any possible moderation is up to the appropriate UPnP Forum working committee (for standard services) or a UPnP vendor (for non-standard services).

4.5 Eventing references

FXPP
: Flexible XML Processing Profile. Specifies that unknown XML elements and their sub elements must be ignored. IETF draft.

GENA
: General Event Notification Architecture. IETF draft.

XML Schema (Part 1: Structures, Part 2: Datatypes)
: Grammar defining UPnP Template Language. Defined using XML. W3C working draft. Part 1: Structures. Part 2: Datatypes.

5. Presentation

Presentation is Step 5 in UPnP networking. Presentation comes after addressing (Step 0) where devices get network addresses, after discovery (Step 1) where control points find interesting device(s), and after description (Step 2) where control points learn about device capabilities. Presentation exposes an HTML-based user interface for controlling and/or viewing device status. Presentation is complementary to control (Step 3) where control points send actions to devices, and eventing (Step 4) where control points listen to state changes in device(s).

After a control point has (1) discovered a device and (2) retrieved a description of the device, the control point is ready to begin presentation. If a device has a URL for presentation, then the control point can retrieve a page from this URL, load the page into a browser, and depending on the capabilities of the page, allow a user to control the device and/or view device status. The degree to which each of these can be accomplished depends on the specific capabilities of the presentation page and device.

The URL for presentation is contained within the presentationURL element in the device description. The device description is delivered via a description message. The section on Description explains the device description and description messages in detail.

Retrieving a presentation page is a simple HTTP-based process and uses the following subset of the overall UPnP protocol stack. (The overall UPnP protocol stack is listed at the beginning of this document.)

UPnP vendor [purple]
UPnP Device Architecture [green]
HTTP [black]
TCP [black]
IP [black]

At the highest layer, the presentation page is specified by a UPnP vendor. Moving down the stack, the UPnP Device Architecture specifies that this page be written in HTML. The page is delivered via HTTP over TCP over IP. For reference, colors in [square brackets] are included for consistency with other sections in this document.

To retrieve a presentation page, the control point issues an HTTP GET request to the presentation URL, and the device returns a presentation page.

Unlike the UPnP Device and Service Templates, and standard device and service types, the capabilities of the presentation page are completely specified by the UPnP vendor. The presentation page is not under the auspices of a UPnP Forum working committee. The page must be an HTML page; it should be version HTML 3.0 or later. However, other design aspects are left to the vendor to specify. This includes, but is not limited to, all capabilities of the control point's browser, scripting language or browser plug-ins used, and means of interacting with the device. To implement a presentation page, a UPnP vendor may wish to use UPnP mechanisms for control and/or eventing, leveraging the device's existing capabilities but is not constrained to do so.

ROKU EXH. 1002

Presentation pages should use mechanisms provided by HTML for localization (e.g., META tag with charset attribute). Control points should use the ACCEPT- / CONTENT-LANGUAGE feature of HTTP to try to retrieve a localized presentation page. Specifically, a control point may include a HTTP ACCEPT-LANGUAGE header in the request for a presentation page; if an ACCEPT-LANGUAGE header is present in the request, the response must include a CONTENT-LANGUAGE header to identify the page's language.

5.1 Presentation references

HTML
: HyperText Markup Language. W3C recommendation. <http://www.w3.org/MarkUp/>.

Glossary

action
: Command exposed by a service. Takes one or more input or output arguments. May have a return value. For more information, see sections on Description and Control.

argument
: Parameter for action exposed by a service. May be in xor out. For more information, see sections on Description and Control.

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

device
: Logical device. A container. May embed other logical devices. Embeds one or more services. For more information, see section on Description.

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

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

event
: Notification of one or more changes in state variables exposed by a service. For more information, see section on Eventing.

publisher
: Source of event messages. Typically a device's service. For more information, see section on Eventing.

root device
: A logical device that is not embedded in any other logical device. For more information, see section on Description.

service
: Logical functional unit. Smallest units of control. Exposes actions and models the state of a physical device with state variables. For more information, see section on Control.

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

service type

ROKU EXH. 1002

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

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

SSDP
: Simple Service Discovery Protocol. A multicast discovery and search mechanism that uses a multicast variant of HTTP over UDP. For more information, see section on Discovery.

state variable
: Single facet of a model of a physical service. Exposed by a service. Has a name, data type, optional default value, optional constraints values, and may trigger events when its value changes. For more information, see sections on Description and Control.

subscriber
: Recipient of event messages. Typically a control point. For more information, see section on Eventing.

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

UPnP Service Template
: Template listing action names, parameters for those actions, state variables, and properties of those state variables. Written in XML syntax and derived from the UPnP Template Language. Defined by a UPnP Forum working committee. One-to-one relationship with standard service types. For more information, see section on Description.

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

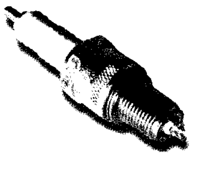
EOF

APPENDIX V

SERVICE LOCATION PROTOCOL: Automatic Discovery of IP Network Services

ERIK GUTTMAN, *Sun Microsystems*

As computers become more portable and networks larger and more pervasive, the need to automate the location and client configuration for network services also increases. The Service Location Protocol is an IETF standard that provides a scalable framework for automatic resource discovery on IP networks.



The complexity of configuring every element in the network—clients, servers, peers, and infrastructure—is a key problem facing network technology's advance. As long as configuration remains difficult, network administration will be expensive, tedious, and troublesome, and users will be unable to take advantage of the full range of capabilities networked systems could provide. The Service Location Protocol[1] is an Internet Engineering Task Force standard for enabling network-based applications to automatically discover the location—including address or domain name and other configuration information—of a required service. Clients can connect to and make use of services using SLP. Currently, without SLP, service locations must be manually configured or entered into a configuration file. SLP provides for fully decentralized operation and scales from small, unadministered networks to large enterprise networks with policies dictating who can discover which resources.

This article describes SLP's operation and how it adapts to conditions where infrastructure is not available, where administration is minimal, or where network administrators simply wish to reduce workload.

BACKGROUND

The Service Location Protocol (SVRLOC) working group has been active in the IETF for several years. In 1997, the group published SLP Version 1 as a Proposed Standard RFC.[1] In June 1999, the Internet Engineering Steering Group announced that Version 2 and its related documents were promoted to Proposed Standard RFCs as well.[2] SLPv2, which updates and replaces SLPv1, is the subject of this article. It removes several of the originally imposed requirements, provides protocol extensibility (new options can be added without modifying the base protocol), adheres to new IESG protocol recommendations, improves security, and eliminates a number of inconsistencies in the SLPv1 specification.

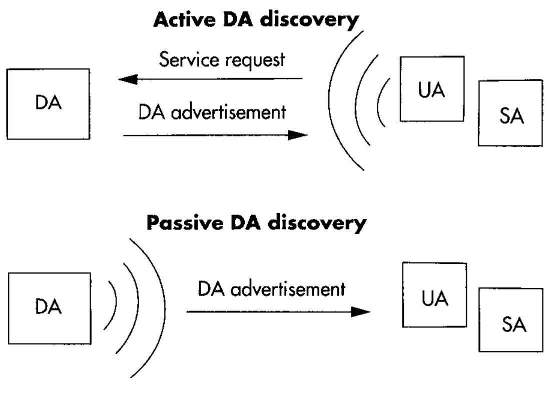


Figure 1. Methods of DA discovery. In active discovery, User Agents and Service Agents multicast requests to locate Directory Agents on the network, whereas, in passive discovery, UAs and SAs learn of DAs via periodic multicast advertisements.

Backward compatibility with SLPv1 depends on whether the Directory Agent (described in the next section) supports both versions. For example, Sun has successfully implemented a backwardly compatible SLP DA. Otherwise, backward compatibility requires a Service Agent (also described below) to implement both versions of the protocol.

Problems with Earlier Protocols

Prior to SLP, service discovery protocols allowed users to discover services only by type. For instance, both Apple and Microsoft offered networking protocols that could discover instances of printers and file servers, and users had to then select from the list to meet their needs. From the beginning, the SVRLOC working group sought a solution that would allow network software to discover services according to their characteristics as well as type. Thus, clients would be able to explicitly discover services that met their requirements, and software could automatically obtain the service location without bothering users.

On the other hand, since services are advertised along with their characteristics, SLP also enables rich user interaction. SLP enables browser operations since the protocol includes a set of directory-like functions. Thus, clients using SLP can browse all the available types of service. These clients may also request the attributes of a class of service, which aids in formulating interactive requests. Finally, SLP makes it possible to look up the attributes of a particular service once it has been discovered.

Another problem with the proprietary protocols was their notorious lack of scalability. The SRVLOC working group sought to correct this problem by minimizing the impact of service discovery on the network. SLP uses multicast and Dynamic Host Configuration Protocol[3] to initialize its scalable service discovery framework without the need for configuring individual SLP agents. SLP can operate in networks ranging from a single LAN to a network under a common administration, also known as an *enterprise network*. These networks can be quite large (potentially tens of thousands of networked devices). Neither multicast discovery nor DHCP scales to the Internet, since these protocols must be configured and administered. Moreover, the Internet lacks a common centralized administration. To the extent that SLP relies on either multicast discovery or DHCP for its own configuration, SLP does not scale to the Internet.

Current SLP Implementations

Sun Microsystems, Novell, IBM, Apple, Axis Communications, Lexmark, Madison River Technologies, and Hewlett-Packard have adopted SLPv1, and, increasingly, SLPv2 for products. There are also two reference implementations of SLPv2 available from http://www.srvloc.org/.

PROTOCOL OVERVIEW

SLP establishes a framework for resource discovery that includes three "agents" that operate on behalf of the network-based software:

- User Agents (UA) perform service discovery on behalf of client software.
- Service Agents (SA) advertise the location and attributes on behalf of services.
- Directory Agents (DA) aggregate service information into what is initially a stateless repository.

Figure 1 illustrates the two different methods for DA discovery: active and passive. In active discovery, UAs and SAs multicast SLP requests to the network. In passive discovery, DAs multicast advertisements for their services and continue to do this periodically in case any UAs or SAs have failed to receive the initial advertisement.

UAs and SAs can also learn the locations of DAs by using the DHCP options for Service Location, SLP DA Option (78).[4] DHCP servers, configured by network administrators, can use DHCP Option 78 to distribute the addresses of DAs to hosts that request them. SLP agents configured in this man-

ROKU EXH. 1002

ner do not require the use of multicast discovery, since this is only used to discover DAs and to discover services in the absence of DAs.

Operational Modes

SLP has two modes of operation:

- When a DA is present, it collects all service information advertised by SAs, and UAs unicast their requests to the DA.
- In the absence of a DA, UAs repeatedly multicast the same request they would have unicast to a DA. SAs listen for these multicast requests and unicast responses to the UA if it has advertised the requested service.

When a DA is present, UAs receive faster responses, SLP uses less network bandwidth, and fewer (or zero) multicast messages are issued.

Aside from unsolicited announcements sent by DAs, all messages in SLP are requests that elicit responses. By default, SLP agents send all messages in UDP datagrams; TCP is used only to send messages that don't fit in a single datagram.

Service Advertisements

Services are advertised using a Service URL, which contains the service's location: the IP address, port number and, depending on the service type, path. Client applications that obtain this URL have all the information they need to connect to the advertised service. The actual protocol the client uses to communicate with the service is independent of SLP.

Service Templates[5]—documents registered with the Internet Assigned Numbers Authority (IANA)—define the attributes associated with service advertisements. Templates specify the attributes, and their default values and interpretation, for a particular service type. SAs advertise services according to attribute definitions in the Service Templates, and UAs issue requests using these same definitions. This ensures interoperablility between vendors because every client will request services using the same vocabulary, and every service will advertise itself using well-known attributes.

OPERATIONS AND SCENARIOS

SLP operates in several different scenarios.

Initialization

At startup, UAs and SAs first determine whether there are any DAs on the network. DA addresses

In SLP, User Agents and Service Agents can use the multicast convergence algorithm to discover Directory Agents. UAs can also use it to issue requests when no DA is present. This algorithm allows SLP agents to receive replies from more responders than they could with standard multicast. Ordinarily in multicast, if there are many responders, a requester is likely to be inundated by the implosion of responses.

The SLP agent attempting to discover services (or DAs) issues an SLP Service Request message using multicast. This may result in one or more unicast response messages. After a wait period, the request is reissued with an appended "previous responders list," including the address of each SLP agent that has already responded. When SAs or DAs receive requests, they first examine the previous responder list. If they discover themselves on the list, they do not respond to the request.

The previous responder list can contain about 60–100 entries before it, combined with the request, becomes too large to fit in the request datagram. Reliable multicast is a notoriously difficult problem, but SLP's multicast convergence algorithm provides a "semi-reliable" multicast transaction.

Figure A illustrates the algorithm's operation as used in DA discovery. When the request is first sent, DAs 1, 2, and 3 reply, but the reply from DA 3 is lost. When the request is retransmitted a second time, DAs 1 and 2 do not respond, but since DA 3 is not yet on the list, it replies again. On the third retransmission no DAs respond since they all finally appear on the previous responders list.

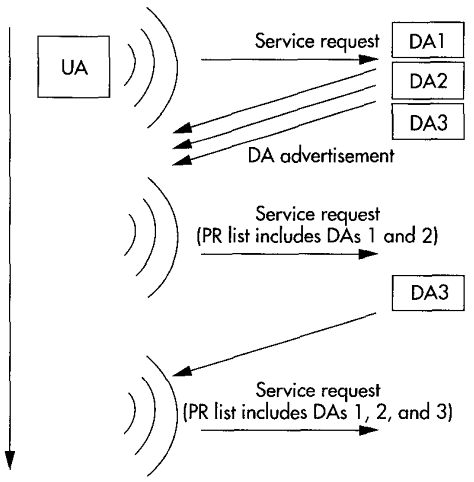


Figure A. The multicast convergence algorithm in DA discovery.

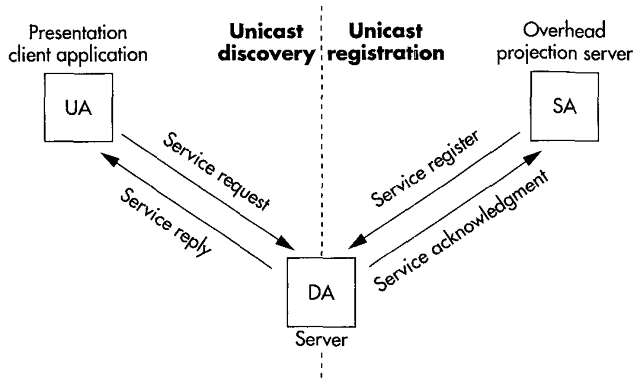


Figure 2. Normal operation of SLP. The SA registers the overhead projection server's location with the DA, and in response to the UA's Service Request message, it obtains this location from the DA via a Service Reply message.

can be configured statically (either manually configured, read from a configuration file, or hard-coded). DA locations can also be obtained dynamically using DHCP as discussed above. In these cases, there is no need to perform DA discovery. In all other cases, UAs and SAs use the SLP multicast convergence algorithm to discover DAs (see the sidebar, "SLP Multicast Convergence Algorithm"). They multicast Service Request messages to obtain DA advertisement messages, which include the Service URL[5] as well as the DA's scope, attributes, and digital signature. From this information UAs and SAs can locate the correct DA for message exchange.

Standard Case

Figure 2 depicts SLP's normal operation. In this example, a client program seeks an overhead projection server to display a presentation to an assembled audience. The SA registers the service's location with the DA, and the UA obtains this location from the DA in a Service Reply message.

Note that the SLP agents depicted may or may not reside on separate networked computers, but only one DA or SA can be on any given machine, due to the rules of the multicast convergence algorithm.

The networked service process advertises itself by registering with a DA, using an internal Service Location Protocol API.[6] An SA on the same computer as the network service registers service information by sending a Service Registration message to the DA and awaiting a Service Acknowledgment in reply.

Service registrations have lifetimes no greater than 18 hours, so the SA must reregister the service periodically, or the lifetime expires. In the event that the service terminates, the SA can optionally send a Service Deregister message to the DA; but even in the worst case, when the service fails, the registration will age out. This ensures that stale information will not persist with the DA.

Client software can use the standard Service Location Protocol API to find the particular service it requires. In this case, a UA sends the DA a Service Request that includes a search filter that is syntactically identical to the request format used by version 3 of the Lightweight Directory Access Protocol.[7] SLP thereby provides a directory-like lookup of all services that match the client's requirements. The DA returns a Service Reply message containing Service URLs and enough information for the client to contact each service that matches the request.

Larger Network Environments

A DA may serve thousands of clients and servers, so SLP gives administrators ways to improve overall performance and scalability of an SLP deployment as DAs become more loaded.

More DAs. First of all, administrators can simply activate more DAs to enhance SLP performance. Because SAs register with each DA they detect, all DAs will eventually contain the same service information, provided that all SAs can find them all. (Of course, sometimes this is neither possible nor desirable.) Adding more DAs creates roughly duplicate repositories of service information without requiring any formal database synchronization between them. Moreover, since UAs can choose any available DA to issue requests to, the load will be shared among DAs. Additional DAs also provide robustness in cases where one fails or becomes overloaded.

Scope. The second mechanism for increasing SLP scalability is scope. A scope is a string used to group resources by location, network, or administrative category. SAs and UAs are by default configured with the scope string "default." Figure 3, for example, shows how a UA issuing requests in the legal department of an organization might find services within that scope, but not in accounts payable. (Note that services may be available in multiple scopes.)

UAs can accumulate DA advertisements to form lists of all available scopes. When no DAs are present, UAs can multicast requests for SA advertisements to create lists of scopes supported by the SAs.

DHCP. SLP agents (UAs, SAs, and DAs) can access non-default scopes via static configuration or

ROKU EXH. 1002

DHCP Option for Service Location, SLP Service Scope Option (79).[4] Because it allows an administrator to easily control the set of services available to a particular client, DHCP is actually the preferred method for configuring SLP. For example, all services and clients in a hotel room could be configured to the scope of that one room. A laptop computer used in a particular room would discover only those services in the room itself. Hotel guests could then locate printers in their own rooms, but not in others to which they have no physical access.

Small Office or Home Networks

When there is no directory agent, a UA multicasts the same requests to the SAs that it would have unicast to the DA. Thus, SAs must be prepared to answer Service Requests. A Service Request includes a query that the SA processes against the attributes of the services it advertises. If the multicast request fails to match, or if the SA is unable to process it (due to an error in the request, for instance), the SA simply discards the request.

The multicast convergence algorithm used for DA discovery is also used by UAs for service discovery.

It is important to note that services can be discovered by clients using SLP in small networks without any SLP-specific configuration or the deployment of any additional services. SLP discovery works even in the absence of DNS, DHCP, SLP DAs, and routing. This makes SLP suitable for the home or small office environment, where impromptu and unadministered networks would greatly benefit from automatic service discovery.

FITTING THE PIECES TOGETHER

SLP, in itself, only provides a service discovery framework. That is, SLP agents are idle until service software is advertised, populating the SAs, which in turn propagate information to appropriate DAs. UAs are inactive until a client issues a specific request.

The SLP API, however, allows applications and services to access SLP's functionality. It provides for both synchronous and asynchronous operations, and it features both C and Java bindings. (Table 1 summarizes the C language bindings, and Figure 4 summarizes the Java language bindings.) The API contains separate interfaces for client software to use for discovery and for server software to use for advertising; peer-to-peer software can use both portions. Figure 5 illustrates the interaction between a client and a service, the SLP API, and a UA and SA performing a Service Request operation.

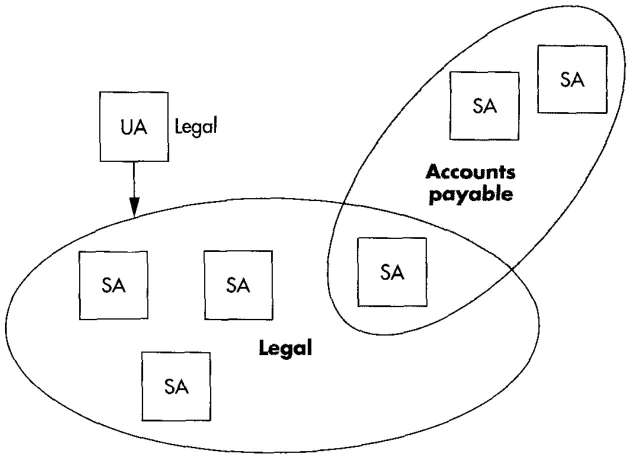


Figure 3. Scope in SLP. User Agents can only locate services within the scopes to which they have access.

```
public interface Advertiser {
  public abstract void register(ServiceURL url, Vector attributes);
  public abstract void addAttributes(ServiceURL url, Vector attributes);
  public abstract void deleteAttributes (ServiceURL url, Vector attributes);
};

public interface Locator {
  public abstract Locale getLocale();
  public abstract ServiceLocationEnumeration
    findServiceTypes(String namingAuthority, Vector scopes)
  public abstract ServiceLocationEnumeration
    findServices(ServiceType type, Vector scopes, String searchFilter)
  public abstract ServiceLocationEnumeration
    findAttributes(ServiceURL URL, Vector scopes, Vector attributeIDs)
  public abstract ServiceLocationEnumeration
    findAttributes(ServiceType type, Vector scopes, Vector attributeIDs)
};

public class ServiceLocationManager {
  public int getRefreshInterval();
  public static Vector findScopes();
  public static Advertiser getAdvertiser();
  public static Locator getLocator();
};
```

Figure 4. SLP API Java bindings.

Details

SLP is a mostly string-based protocol that uses a binary message header, as shown in Figure 6. Messages are largely composed of UTF-8 strings[8] preceded by length fields. (See Table 2 on page 79 for a description of other fields.)

The SLP header concludes with a Language Tag.[9] Attribute value strings in the SLP message are translated into the language indicated by the tag, and the Service Template[5] associated with a partic-

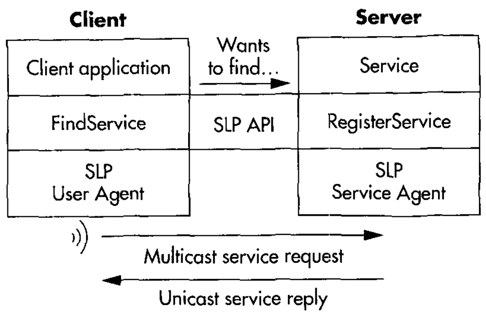


Figure 5. Client server discovery using SLP without DAs. The client application uses a UA to multicast a Service Request that the SA responds to with a unicast Service Reply.

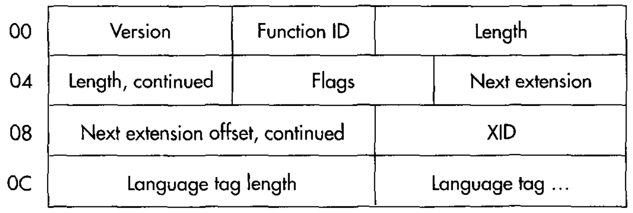


Figure 6. The SLP Header. The SLP header precedes and characterizes all transmitted SLP messages.

ular service provides additional information regarding internationalization. Some strings have standardized translations, while others have fixed meanings and are not intended to be translated.

The Function ID field indicates the SLP message type, all of which are summarized in Table 3 (page 79).

SLP's central function is the exchange of Service Request and Service Reply messages. A UA, for instance, only has to be able to send Service Requests (though it also needs to handle Service Replies and DA advertisements as a result of those requests). An SA need not support any features besides discovering DAs, responding to Service Requests, and sending Service Registration messages to appropriate DAs.

Deployment

Currently, service location information is acquired by prompting the user or reading it from a configuration file. To use SLP, client software vendors must modify the network configuration portions of the client to employ the SLP API[6] to obtain the location of the server. By modifying the failure notification path as well, automatic service discovery can be reinitiated when a server fails. In this way, another server can be located without interrupting the user's service.

Alternatively, it is possible to utilize the benefits of SLP without modifying the client software, as long as the client already uses an existing service for configuration. For instance, some clients use the LDAPv3 protocol[10] to access a directory for configuration information, so they can discover services that SLP has automatically registered with the directory.

SLP attributes, Service Templates, and search filters are all compatible with a subset of LDAPv3. This means that services registered with an SLP DA can be automatically registered into an LDAPv3 directory. That is, an LDAPv3 directory can function as the back end for an SLP DA. Thus, users of the LDAPv3 directory can obtain current network service information automatically. SLP's interoperability with LDAPv3 eases the integration of network configuration information in IP networks, where directories are increasingly used to centralize access.

Table 1. SLP API C language bindings.

Binding	Description
SLPOpen	Clients and services initialize the SLP library and obtain a handle to use with all subsequent calls.
SLPClose	Clients and services release the SLP library.
SLPReg	Services register their service URL and attributes. A service may also use this interface to update its service attributes or refresh a registration before it expires.
SLPDereg	Services can deregister their availability.
SLPDelAttrs	Services can deregister a particular attribute.
SLPFindSrvs	Clients can obtain service URLs based on their query by service type, SLP scope, and/or service attributes. These URLs will, by definition, be the locations of services the client can use.
SLPFindSrvTypes	Clients can discover all types of service available on the network.
SLPFindAttrs	Clients can discover attributes of a particular service or the attributes of all services of a given service type.

AUTOMATIC SER

There is a great deal of work going on in the area of autoconfiguration. Comparison with other approaches shows SLP's generality and versatility.

DHCP Service Options

The location of several types of service can be configured using DHCP. Administrators can configure certain clients for a particular server. For example, DHCP option 42 configures a host to use a particular Network Time Protocol (NTP)[1] server or servers.

Using DHCP to configure services is significantly different from using SLP. DHCP servers, for instance, have no intrinsic way to determine whether an address actually refers to a currently available server. SLP, on the other hand, lets you discover servers with known availability. SLP also allows a client to discover a server that meets its specific requirements. DHCP provides no such mechanism.

DNS Resource Records for Specifying the Location of Services

The Domain Name System (DNS) SRV Resource Record (SRV RR)[2,3] allows for lookups of domain names associated with service names. Thus, a DNS resolver can request all instances of a particular service type within a given domain.

For example, to find all instances of TCP-based line printer (LPR) services in the domain "nonexistent.net," a DNS resolver would send a request to the DNS SRV RR for the service named "lpr.tcp.nonexistent.net." This might return two domain names: "big.nonexistent.net" and "small.nonexistent.net."

Unlike operations in SLP, resolving a DNS SRV RR currently requires a DNS server to be present. Woodcock and Manning currently have an effort underway to standardize a new mechanism to allow multicasting of DNS requests.[4] According to this proposal, individual systems could contain "stub" DNS servers that would respond to multicast requests in the absence of a true DNS server. This mechanism would remove the requirement for a DNS server to be present, making the DNS SRV RR's approach suitable for small networks lacking in administration and infrastructure.

This service discovery method, however, allows the client to discover services only by type, and not by service characteristics.

There is currently no way to update DNS servers when services become available or go down; as with DHCP, the client system might easily obtain locations for services that are not currently available.

Simple Multicast Discovery Protocols

A variety of simple multicast discovery protocols have been proposed over the years.[5-7] In all of them, a client multicasts a request for a desired service type. All available services that receive the multicast request send a response, including the location of services matching the type requested.

Some of the proposals allow services to announce their presence as they come up and periodically thereafter, so clients can become immediately aware of new services. However, none of them scales beyond a small network. Unlike SLP, they provide no means for automatic service information collection. Moreover, they cannot detect that they are in a larger network and should stop multicasting or limit their TTL to avoid disrupting operations.

Finally, these protocols fail to include any features for reliable multicasting. If dozens of responders attempt to reply to a multicast request, the implosion of replies may inundate the requester. SLP ameliorates this problem by using previous-

continued on p. 78

ADDITIONAL FEATURES

The essential function of SLP is service discovery. SLP has also been designed to provide security, extensibility, support for browsing operations, and operation over IPv6. These features extend the utility of SLP, and will be especially useful once a standardized security infrastructure has been widely adopted on IP networks.

Security

SLP is designed to make service information available, and it contains no mechanisms to restrict access to this information. Its only security property is authentication of the source of information, which prevents SLP from being used to maliciously propagate false information about the location of services.

Digital signatures. An SA can include a digital signature produced with public key cryptography along with its registration messages. A DA can then verify the signature before registering or deregistering any service information on the SA's behalf. These digital signatures are then forwarded in reply messages to UAs, so they can reject unsigned or incorrectly signed service information. Of course, DAs and UAs can only verify signatures, not produce them.

continued from p. 77

responder lists in its multicast convergence algorithm (as discussed in the sidebar). Of course, the replies could be staggered by requiring responders to wait for a random interval, but this would force the requester to wait much longer for answers where there are few results.

Jini

Sun Microsystems' Jini technology provides a Java-oriented set of mechanisms and programmer interfaces for automatic configuration. As an IETF standard, SLP is a general mechanism suited to heterogeneous systems. Jini, on the other hand, leverages Java's uniformity across platforms, providing powerful semantics for service discovery operations.

The Jini discovery architecture is similar to that of SLP. Jini agents discover the existence of a Jini Lookup Server, which collects service advertisements in a manner analogous to DAs in SLP. Jini agents then request services on behalf of client software by communicating with the Lookup Server. Unlike SLP, however, where DAs are optional, Jini requires the presence of one or more Lookup Servers.

Jini's discovery mechanism offers some advantages to Java-based clients. The Lookup Server uses object-oriented matching to determine which services support the client's requested Java interface. Both Jini and SLP use attributes to find services that match the client's requirements, but where SLP uses string-based attributes and weak typing, Jini employs Java objects throughout.

Service discovery with SLP returns a URL denoting a service's location. Jini, on the other hand, returns an object that offers direct access to the service, using an interface known to the client.

For some embedded systems that offer network services, running a Java Virtual Machine may require memory and processing resources that are too costly, thus precluding the use of Jini to advertise services. In those cases, SLP can advertise the services as well as a "Java Driver Factory." A Java Driver Factory is a class that can be used to produce (instantiate and initialize) Java objects based upon initialization parameters. An SLP-Jini bridge can detect services and obtain their attributes and Java Driver Factory. (For more, see http://www.srvloc.org) The bridge uses the Java Driver Factory to instantiate a Java driver object, initializing it with the attributes advertised using SLP. The bridge then registers the service with a Jini Lookup server.

When a Jini client discovers a service, it will be able to use it equally well, regardless of whether it was directly registered with the Jini Lookup Server or registered by proxy via an SLP-Jini bridge. The bridge allows clients to make use of Jini's powerful API to discover services on the network that cannot support Jini natively themselves.

REFERENCES

1. D. Mills, "Network Time Protocol (Version 3). Specification, Implementation and Analysis," RFC 1305, Mar. 1992; available at http://www.rfc-editor.org/rfc/rfc1305.txt.
2. A. Gulbrandsen and P. Vixie, "A DNS RR for Specifying the Location of Services (DNS SRV)," RFC 2052, Oct. 1996; available at http://www.rfc-editor.org/rfc/rfc2052.txt.
3. P. Mockapetris, "Domain Names—Implementation and Specification," RFC 1035, Nov. 1987; available at http://www.rfc-editor.org/rfc/rfc1035.txt.
4. B. Woodcock and B. Manning, "Multicast Discovery of DNS Services," Dec. 1998, work in progress.
5. D. Brown, "IPLookup Service," *Personal Communication*, 19 June 1997.
6. S. Hanton, "Simple Server Discovery Protocol," Jan. 1997, work in progress.
7. T. Cai, et al., "Simple Service Discovery Protocol/1.0," Apr. 1999, work in progress.

As an additional level of authentication, DAs can also include digital signatures with their advertisements. UAs and SAs can thus avoid DAs that have not been legitimately established by the site's administration because SLP agents that possess private keys for generating verifiable digital signatures are (by definition) trusted to legitimately advertise themselves.

Security Configuration Requirements. SLP is designed to automatically configure service locations with minimal static configuration requirements for SLP agents. SLP security, however, does require some additional configuration (for the cryptographic keys or certificates used in generating and verifying digital signatures).

When needed, vendors of SLP-enabled clients and services can establish new cryptographic algorithms and data formats within SLP's existing protocol. It is also possible to deploy new keys gradually, without requiring flag days, which would require simultaneous reconfiguration of all interoperating systems. Suppose, for example, that corporate policy requires that old private keys for authenticating servers be replaced in all SAs in the enterprise every three months. If flag days were required, all SAs, UAs, and DAs would have to be rekeyed at once. SLP, on the other hand, allows phasing in of new keys. SAs include digital signatures generated by both the new and old keys with their messages until all UAs and DAs replace the old keys, at which point, they phase out the old generation of keys.

Extensibility

SLP extensions are additional protocol elements appended to messages. For example, there is an extension for reporting when a service request omits an attribute that is defined as required by a Service Template.[5] Another extension currently under development would allow UAs to request notification of additional services as they appear on the network.

When an SLP agent recognizes a message extension, it will perform the appropriate processing. If the extension is not recognized, it is either ignored or the entire SLP message is discarded, depending on how the message extension is labeled.

SLP's extensibility allows for future enhancements—such as additional error reporting, added notification facility, and so on—without altering the base protocol.

Browsing Features

In addition to its required features, the SLP specification describes several optional features that could be used to support sophisticated service browsers.

Service Type Request. Using this type of message, UAs can discover all service types available on the network. The response supplies a top-level taxonomy of services, which supports the basic requirements for building a general "service browser" on top of SLP.

Attribute Request. A UA can use the Attribute Request to retrieve all the attributes of a given service in a manner similar to a directory lookup operation. The UA can also issue the request *without* naming a specific service instance. The response from the DA (or SAs if there is no DA) returns all attributes and values for the requested service type within the network. For example, an Attribute Request for all available video servers on the network might return the following properties:

- locations include my building and a remote office;
- video streams include programs A and B.

Table 2. SLP header fields.

Header Field	Description
Version	SLP protocol version: 1 and 2 are defined.
Length	Length of the entire SLP message.
Function ID	Message type that follows the SLP Header.
Flags	Indicate special treatment of the message.
Next Extension Offset	Offset, in bytes, to the first SLP Extension.
XID	Unique number for each unique request.
Language Tag Length	Length of the Language Tag that follows.
Language Tag	Indicates the language of all human-readable strings included in the SLP message.

Table 3. SLP message types and descriptions.

SLP Message Type	ID	Description
Service Request	1	UAs find service by type, scope, and search filter.
Service Reply	2	DA (or SA) returns Service URLs and their lifetimes.
Service Register	3	SAs register Service URLs and attributes.
Service Deregister	4	SAs deregister Service URLs and attributes.
Service Acknowledgment	5	DAs acknowledge a successful registration or deregistration.
Attribute Request	6	UAs find attributes by service type or by Service URL.
Attribute Reply	7	DA (or SA) returns attribute information.
DAAdvert	8	DA sends its Service URL, scope, and attributes.
Service Type Request	9	UAs find service types by scope.
Service Type Reply	10	DA (or SA) returns a list of service types.
SAAdvert	11	SA sends its Service URL, scope, and attributes.

With this information, a browser interface could help me determine the location of a video server in my building or a video server that serves video stream A. If, however, I request a server that is in my building and serves video stream A, I might not succeed because the attributes are independent: video stream A might only be available from the server in the remote office.

Attribute Request messages are also useful for determining the attributes of a particular service. The UA locates the corresponding Service URL, and the Attribute Request uses it to look up the service's attributes. Network software could also discover a particular service's features by requesting the attributes directly, which would require subsequent protocol feature negotiation between client and server.

SLP Operation over IPv6

The formal specification has not yet been standardized, but SLP is designed to provide service discovery facilities that will work for networks using IPv6.[11] Once the debate is settled regarding which string representation to use in URLs for numerical IPv6 addresses, some minor changes will be needed. Service URLs[5] containing numerical addresses will require a different format from what IPv4 uses, and link-local addresses will require some special handling in IPv6. For example, DAs that obtain service registrations with link-local numerical addresses must not forward them using the link on which they were registered. Also, the address to use for site-local scoped multicast operations differs in IPv4 from what it is in IPv6.[12]

SUMMARY

SLP is an IETF standard for service discovery and automatic configuration of clients. It provides for fully decentralized operation and scales from a small, unadministered network to an enterprise network where policy may dictate who should discover which resources. This paper describes how SLP operates and how it adapts to conditions where infrastructure is not available, where administration is minimized, and where network administrators in large enterprises wish to reduce tedium and workload. While alternative mechanisms exist, SLP remains the most general and versatile solution for service discovery on TCP/IP networks. ■

REFERENCES

1. J. Veizades, E. Guttman, and C. Perkins, "Service Location Protocol," IETF, RFC 2165, June 1997; available at http://www.rfc-editor.org/rfc/rfc2165.txt.
2. E. Guttman, C. Perkins, J. Veizades, and M. Day, "Service Location Protocol, Version 2," IETF, RFC 2608, June 1999; available at http://www.rfc-editor.org/rfc/rfc2608.txt.
3. R. Droms, "Dynamic Host Configuration Protocol," IETF, RFC 2131, Mar. 1997; available at http://www.rfc-editor.org/rfc/rfc2131.txt.
4. C. Perkins and E. Guttman, "DHCP Options for Service Location Protocol," IETF, RFC 2610, June 1999; available at http://www.rfc-editor.org/rfc/rfc2610.txt.
5. E. Guttman, C. Perkins, and J. Kempf, "Service Templates and Service: Schemes," IETF, RFC 2609, June 1999; available at http://www.rfc-editor.org/rfc/rfc2609.txt.
6. J. Kempf and E. Guttman, "An API for Service Location," IETF, RFC 2614, June 1999; available at http://www.rfc-editor.org/rfc/rfc2614.txt.
7. T. Howes, "The String Representation of LDAP Search Filters." IETF, RFC 2254, Dec. 1997; available at http://www.rfc-editor.org/rfc/rfc2254.txt.
8. F. Yergeau, "UTF-8, a Transformation Format of ISO 10646," IETF, RFC 2279, Jan. 1998; available at http://www.rfc-editor.org/rfc/rfc2279.txt.
9. H. Alvestrand, "Tags for the Identification of Languages," IETF, RFC 1766, Mar. 1995; available at http://www.rfc-editor.org/rfc/rfc1766.txt.
10. M. Wahl, T. Howes, and S. Kille, "Lightweight Directory Access Protocol, version 3," IETF, RFC 2251, Dec. 1997; available at http://www.rfc-editor.org/rfc/rfc2251.txt.
11. E. Guttman and J. Veizades, "Service Location Protocol Modifications for IPv6," Oct 1998, work in progress.
12. R. Hinden and S. Deering, "IP Version 6 Multicast Address Assignments," IETF, RFC 2375, July 1998; available at http://www.rfc-editor.org/rfc/rfc2375.txt.

FURTHER READING ON SLP

J. Kempf and P. St.Pierre, *Service Location Protocol for Enterprise Networks*, John Wiley & Sons, 1999.

Service Location Protocol Home Page • http://www.svrloc.org/.

Erik Guttman is a staff engineer at Sun Microsystems. He is a member of the Advanced Network Development team in Sun Labs. His technical interests include automatic configuration, network security, and network software testing. He is an active member of the Internet Engineering Task Force where he is the chairman of the Service Location Protocol (SVRLOC) Working Group. He received a BA in Philosophy and Computer Science from UC Berkeley and an MS in Computer Science from Stanford University.

Readers can contact Erik Guttman at erik.guttman@sun.com.

ROKU EXH. 1002

APPENDIX W

Network Working Group
Request for Comments: 2165
Category: Standards Track

J. Veizades
@Home Network
E. Guttman
C. Perkins
Sun Microsystems
S. Kaplan

June 1997

Service Location Protocol

Status of This Memo

Abstract

The Service Location Protocol provides a scalable framework for the discovery and selection of network services. Using this protocol, computers using the Internet no longer need so much static configuration of network services for network based applications. This is especially important as computers become more portable, and users less tolerant or able to fulfill the demands of network system administration.

Table of Contents

1. Introduction 3
2. Terminology 3
   2.1. Notation Conventions 5
   2.2. Service Information and Predicate Representation 5
   2.3. Specification Language 6
3. Protocol Overview 6
   3.1. Protocol Transactions 7
   3.2. Schemes 8
      3.2.1. The "service:" URL scheme 9
   3.3. Standard Attribute Definitions 9
   3.4. Naming Authority 10
   3.5. Interpretation of Service Location Replies 10
   3.6. Use of TCP, UDP and Multicast in Service Location 10
      3.6.1. Multicast vs. Broadcast 11
      3.6.2. Service-Specific Multicast Address 11
   3.7. Service Location Scaling, and Multicast Operating Modes . 12

```
 4. Service Location General Message Format                          14
     4.1. Use of Transaction IDs (XIDs) . . . . . . . . . . . . . . .  15
     4.2. URL Entries . . . . . . . . . . . . . . . . . . . . . . . .  16
     4.3. Authentication Blocks . . . . . . . . . . . . . . . . . . .  17
     4.4. URL Entry Lifetime  . . . . . . . . . . . . . . . . . . . .  19
 5. Service Request Message Format                                   19
     5.1. Service Request Usage . . . . . . . . . . . . . . . . . . .  22
     5.2. Directory Agent Discovery Request . . . . . . . . . . . . .  23
     5.3. Explanation of Terms of Predicate Grammar . . . . . . . . .  24
     5.4. Service Request Predicate Grammar . . . . . . . . . . . . .  26
     5.5. String Matching for Requests  . . . . . . . . . . . . . . .  27
 6. Service Reply Message Format                                     28
 7. Service Type Request Message Format                              29
 8. Service Type Reply Message Format                                31
 9. Service Registration Message Format                              32
10. Service Acknowledgement Message Format                           35
11. Service Deregister Message Format                                37
12. Attribute Request Message Format                                 38
13. Attribute Reply Message Format                                   40
14. Directory Agent Advertisement Message Format                     42
15. Directory Agents                                                 43
    15.1. Introduction  . . . . . . . . . . . . . . . . . . . . . . .  43
    15.2. Finding Directory Agents  . . . . . . . . . . . . . . . . .  43
16. Scope Discovery and Use                                          45
    16.1. Protected Scopes  . . . . . . . . . . . . . . . . . . . . .  46
17. Language and Character Encoding Issues                           47
    17.1. Character Encoding and String Issues  . . . . . . . . . . .  48
          17.1.1. Substitution of Character Escape Sequences  . . . .  49
    17.2. Language-Independent Strings  . . . . . . . . . . . . . . .  49
18. Service Location Transactions                                    50
    18.1. Service Location Connections  . . . . . . . . . . . . . . .  50
    18.2. No Synchronous Assumption . . . . . . . . . . . . . . . . .  51
    18.3. Idempotency . . . . . . . . . . . . . . . . . . . . . . . .  51
19. Security Considerations                                          51
20. String Formats used with Service Location Messages               52
    20.1. Previous Responders' Address Specification  . . . . . . . .  53
    20.2. Formal Definition of the "service:" Scheme  . . . . . . . .  53
          20.2.1. Service Type String . . . . . . . . . . . . . . . .  54
    20.3. Attribute Information . . . . . . . . . . . . . . . . . . .  54
    20.4. Address Specification in Service Location . . . . . . . . .  55
    20.5. Attribute Value encoding rules  . . . . . . . . . . . . . .  55
21. Protocol Requirements                                            56
    21.1. User Agent Requirements . . . . . . . . . . . . . . . . . .  56
    21.2. Service Agent Requirements  . . . . . . . . . . . . . . . .  58
    21.3. Directory Agent Requirements  . . . . . . . . . . . . . . .  59
22. Configurable Parameters and Default Values                       61
    22.1. Service Agent:  Use Predefined Directory Agent(s) . . . . .  62
    22.2. Time Out Intervals  . . . . . . . . . . . . . . . . . . . .  63
```

23. Non-configurable Parameters 63
24. Acknowledgments 64
A. Appendix: Technical contents of ISO 639:1988 (E/F): "Code for the representation of names of languages" 65
B. SLP Certificates 66
C. Example of deploying SLP security using MD5 and RSA 68
D. Example of use of SLP Certificates by mobile nodes 68
E. Appendix: For Further Reading 69

1. Introduction

Traditionally, users find services by using the name of a network host (a human readable text string) which is an alias for a network address. The Service Location Protocol eliminates the need for a user to know the name of a network host supporting a service. Rather, the user names the service and supplies a set of attributes which describe the service. The Service Location Protocol allows the user to bind this description to the network address of the service.

Service Location provides a dynamic configuration mechanism for applications in local area networks. It is not a global resolution system for the entire Internet; rather it is intended to serve enterprise networks with shared services. Applications are modeled as clients that need to find servers attached to the enterprise network at a possibly distant location. For cases where there are many different clients and/or services available, the protocol is adapted to make use of nearby Directory Agents that offer a centralized repository for advertised services.

2. Terminology

User Agent (UA)
A process working on the user’s behalf to acquire service attributes and configuration. The User Agent retrieves service information from the Service Agents or Directory Agents.

Service Agent (SA)
A process working on the behalf of one or more services to advertise service attributes and configuration.

Service Information
A collection of attributes and configuration information associated with a single service. The Service Agents advertise service information for a collection of service instances.

```
   Service   The service is a process or system providing a facility
             to the network.  The service itself is accessed using a
             communication mechanism external to the the Service
             Location Protocol.

   Directory Agent (DA)
             A process which collects information from Service Agents
             to provide a single repository of service information in
             order to centralize it for efficient access by User
             Agents.  There can only be one DA present per given
             host.

   Service Type
             Each type of service has a unique Service Type string.
             The Service Type defines a template, called a "service
             scheme", including expected attributes, values and
             protocol behavior.

   Naming Authority
             The agency or group which catalogues given Service Types
             and Attributes.  The default Naming Authority is IANA,
             the Internet Assigned Numbers Authority.

   Keyword
             A string describing a characteristic of a service.

   Attribute
             A (class, value-list) pair of strings describing a
             characteristic of a service.  The value string may be
             interpreted as a boolean, integer or opaque value if it
             takes specific forms (see section 20.5).

   Predicate
             A boolean expression of attributes, relations and
             logical operators.  The predicate is used to find
             services which satisfy particular requirements.  See
             section 5.3.

   Alphanumeric
             A character within the range ’a’ to ’z’, ’A’ to ’Z’, or

   Scope     A collection of services that make up a logical group.
             See sections 3.7 and 16.
```

```
      Site Network
                All the hosts accessible within the Agent's multicast
                radius, which defaults to a value appropriate for
                reaching all hosts within a site (see section 22).  If
                the site does not support multicast, the agent's site
                network is restricted to a single subnet.

      URL       A Universal Resource Locator - see [6].

      Address Specification
                This is the network layer protocol dependent mechanism
                for specifying an Agent.  For Internet systems this is
                part of a URL.
```

2.1. Notation Conventions

```
      CAPS   Strings which appear in all capital letters are protocol
             literal.  All string comparison is case insensitive,
             however, (see section 5.5).  Some strings are quoted in
             this document to indicate they should be used literally.
             Single characters inside apostrophes are included
             literally.

      <>     Values set off in this manner are fully described in
             section 20.  In general, all definitions of items in
             messages are described in section 20 or immediately
             following their first use.

      |  |
      \  \   Message layouts with this notation indicate a variable
      |  |   length field.
```

2.2. Service Information and Predicate Representation

Service information is represented in a text format. The goal is that the format be human readable and transmissible via email. The location of network services is encoded as a Universal Resource Locator (URL) which is human readable. Only the datagram headers are encoded in a form which is not human readable. Strings used in the Service Location Protocol are NOT null-terminated.

Predicates are expressed in a simple boolean notation using keywords, attributes, and logical connectives, as described in Section 5.4.

The logical connectives and subexpressions are presented in prefix-order, so that the connective comes first and the expressions it operates on follow afterwards.

2.3. Specification Language

In this document, several words are used to signify the requirements of the specification [8]. These words are often capitalized.

MUST	This word, or the adjective "required", means that the definition is an absolute requirement of the specification.
MUST NOT	This phrase means that the definition is an absolute prohibition of the specification.
SHOULD	This word, or the adjective "recommended", means that, in some circumstances, valid reasons may exist to ignore this item, but the full implications must be understood and carefully weighed before choosing a different course. Unexpected results may result otherwise.
MAY	This word, or the adjective "optional", means that this item is one of an allowed set of alternatives. An implementation which does not include this option MUST be prepared to interoperate with another implementation which does include the option.
silently discard	The implementation discards the datagram without further processing, and without indicating an error to the sender. The implementation SHOULD provide the capability of logging the error, including the contents of the discarded datagram, and SHOULD record the event in a statistics counter.

3. Protocol Overview

The basic operation in Service Location is that a client attempts to discover the location of a Service. In smaller installations, each service will be configured to respond individually to each client. In larger installations, services will register their services with one or more Directory Agents, and clients will contact the Directory Agent to fulfill requests for Service Location information. Clients may discover the whereabouts of a Directory Agent by preconfiguration, DHCP [2, 11], or by issuing queries to the Directory Agent Discovery multicast address.

3.1. Protocol Transactions

The diagram below illustrates the relationships described below:

```
      +---------------+   we want this info:     +-----------+
      |  Application  | - - - - - - - - - - - -> |  Service  |
      +---------------+                          +-----------+
           /|\                                      |     |
            |                         +-------------+     |
            |                         |                   |
           \|/                       \|/                 \|/
      +---------------+          +-----------+      +----------------+
      |   User Agent  |<-------->|  Service  |      |    Service     |
      +---------------+          |   Agent   |      | Agent which    |
            |                    +-----------+      | does not reply |
            |                         |             | to UA requests |
            |                       \|/            +----------------+
            |                   +-------------+           |
            +------------------>|  Directory  |<----------+
                                |    Agent    |
                                +-------------+      ___________
                                     /|\            / Many other\
                                      +------------>|   SA's    |
                                                    \___________/
```

The following describes the operations a User Agent would employ to find services on the site’s network. The User Agent needs no configuration to begin network interaction. The User Agent can acquire information to construct predicates which describe the services that match the user’s needs. The User Agent may build on the information received in earlier network requests to find the Service Agents advertising service information.

A User Agent will operate two ways: If the User Agent has already obtained the location of a Directory Agent, the User Agent will unicast a request to it in order to resolve a particular request. The Directory Agent will unicast a reply to the User Agent. The User Agent will retry a request to a Directory Agent until it gets a reply, so if the Directory Agent cannot service the request (say it has no information) it must return an response with zero values, possibly with an error code set.

If the User Agent does not have knowledge of a Directory Agent or if there are no Directory Agents available on the site network, a second mode of discovery may be used. The User Agent multicasts a request to the service-specific multicast address, to which the service it wishes to locate will respond. All the Service Agents which are listening to this multicast address will respond, provided they can satisfy the User Agent's request. A similar mechanism is used for Directory Agent discovery; see section 5.2. Service Agents which have no information for the User Agent MUST NOT respond.

When a User Agent wishes to obtain an enumeration of ALL services which satisfy the query, a retransmission/convergence algorithm is used. The User Agent resends the request, together with a list of previous responders. Only those Service Agents which are not on the list respond. Once there are no new responses to the request the accumulation of responses is deemed complete. Depending on the length of the request, around 60 previous responders may be listed in a single datagram. If there are more responders than this, the scaling mechanisms described in section 3.7 should be used.

While the multicast/convergence model may be important for discovering services (such as Directory Agents) it is the exception rather than the rule. Once a User Agent knows of the location of a Directory Agent, it will use a unicast request/response transaction.

The Service Agent SHOULD listen for multicast requests on the service-specific multicast address, and MUST register with an available Directory Agent. This Directory Agent will resolve requests from User Agents which are unicasted using TCP or UDP. This means that a Directory Agent must first be discovered, using DHCP, the DA Discovery Multicast address, the multicast mechanism described above, or manual configuration. See section 5.2.

A Service Agent which does not respond to multicast requests will not be useful in the absence of Directory Agents. Some Service Agents may not include this functionality, if an especially lightweight implementation is required.

If the service is to become unavailable, it should be deregistered with the Directory Agent. The Directory Agent responds with an acknowledgment to either a registration or deregistration. Service Registrations include a lifetime, and will eventually expire. Service Registrations need to be refreshed by the Service Agent before their Lifetime runs out. If need be, Service Agents can advertise signed URLs to prove that they are authorized to provide the service.

3.2. Schemes

The Service Location Protocol, designed as a way for clients to access resources on the network, is a natural application for Universal Resource Locators (URLs). It is intended that by re-using URL specification and technology from the World Wide Web, clients and servers will be more flexible and able to be written using already existing code. Moreover, it is hoped that browsers will be written to take advantage of the similarity in locator format, so that a client can dynamically formulate requests for services that are resolved differently depending upon the circumstances.

3.2.1. The "service:" URL scheme

The service URL scheme is used by Service Location. It is used to specify a Service Location. Many Service Types will be named by including a scheme name after the "service:" scheme name. Service Types are used by SAs to register and deregister Services with DAs. It is also used by SAs and DAs to return Service Replies to UAs. The formal definition of the "service:" URL scheme is in section 20.2. The format of the information which follows the "service:" scheme should as closely as possible follow the URL structure and semantics as formalized by the IETF standardization process.

Well known Service Types are registered with the IANA and templates are available as RFCs. Private Service Types may also be supported.

3.3. Standard Attribute Definitions

Service Types used with the Service Location Protocol must describe the following:

Service Type string of the service
Attributes and Keywords
Attribute Descriptions and interpretations

Service Types not registered with IANA will use their own Naming Authority string. The registration process for new Service Types is defined in [13].

Services which advertise a particular Service Type must support the complete set of standardized attributes. They may support additional attributes, beyond the standardized set. Unrecognized attributes MUST be ignored by User Agents.

Service Type names which begin with "x-" are guaranteed not to conflict with any officially registered Service Type names. It is suggested that this prefix be used for experimental or private Service Type names. Similarly, attribute names which begin with "x-" are guaranteed not to be used for any officially registered attribute names.

A service of a given Service Type should accept the networking protocol which is implied in its definition. If a Service Type can accept multiple protocols, configuration information SHOULD be included in the Service Type attribute information. This configuration information will enable an application to use the results of a Service Request and Attribute Request to directly connect to a service.

See section 20.2.1 for the format of a Service Type String as used in the Service Location Protocol.

3.4. Naming Authority

The Naming Authority of a service defines the meaning of the Service Types and attributes registered with and provided by Service Location. The Naming Authority itself is a string which uniquely identifies an organization. If no string is provided IANA is the default. IANA stands for the Internet Assigned Numbers Authority.

Naming Authorities may define Service Types which are experimental, proprietary or for private use. The procedure to use is to create a 'unique' Naming Authority string and then specify the Standard Attribute Definitions as described above. This Naming Authority will accompany registration and queries, as described in sections 5 and 9.

3.5. Interpretation of Service Location Replies

Replies should be considered to be valid at the time of delivery. The service may, however, fail or change between the time of the reply and the moment an application seeks to make use of the service. The application making use of Service Location MUST be prepared for the possibility that the service information provided is either stale or incomplete. In the case where the service information provided does not allow a User Agent to connect to a service as desired, the Service Request and/or Attribute Request may be resubmitted.

Service specific configuration information (such as which protocol to use) should be included as attribute information in Service Registrations. These configuration attributes will be used by applications which interpret the Service Location Reply.

3.6. Use of TCP, UDP and Multicast in Service Location

The Service Location Protocol requires the implementation of UDP (connectionless) and TCP (connection oriented) transport protocols. The latter is used for bulk transfer, only when necessary. Connections are always initiated by an agent request or registration, not by a replying Directory Agent. Service Agents and User Agents use ephemeral ports for transmitting information to the service location port, which is 427.

The Service Location discovery mechanisms typically multicast messages to as many enterprise networks as needed to establish service availability. The protocol will operate in a broadcast environment with limitations detailed in section 3.6.1.

3.6.1. Multicast vs. Broadcast

The Service Location Protocol was designed for use in networks where DHCP is available, or multicast is supported at the network layer. To support this protocol when only network layer broadcast is supported, the following procedures may be followed.

3.6.1.1. Single Subnet

If a network is not connected to any other networks simple network layer broadcasts will work in place of multicast.

Service Agents SHOULD and Directory Agents MUST listen for broadcast Service Location request messages to the Service Location port. This allows UAs which lack multicast capabilities to still make use of Service Location on a single subnet.

3.6.1.2. Multiple Subnets

The Directory Agent provides a central clearing house of information for User Agents. If the network is designed so that a Directory Agent address is statically configured with each User Agent and Service Agent, the Directory Agent will act as a bridge for information that resides on different subnets. The Directory Agent address can be dynamically configured with Agents using DHCP. The address can also be determined by static configuration.

As dynamic discovery is not feasible in a broadcast environment with multiple subnets and manual configuration is difficult, deploying DAs to serve enterprises with multiple subnets will require use of multicast discovery with multiple hops (i.e., TTL > 1 in the IP header).

3.6.2. Service-Specific Multicast Address

This mechanism is used so that the number of datagrams any one service agent receives is minimized. The Service Location General Multicast Address MAY be used to query for any service, though one SHOULD use the service-specific multicast address if it exists.

If the site network does not support multicast then the query SHOULD be broadcast to the Service Location port. If, on the other hand, the underlying hardware will not support the number of needed multicast addresses the Service Location General Multicast Address MAY be used. Service Agents MUST listen on this multicast address as well as the service-specific multicast addresses for the service types they advertise.

Service-Specific Multicast Addresses are computed by calculating a string hash on the Service Type string. The Service Type string MUST first be converted to an ASCII string from whatever character set it is represented in, so the hash will have well-defined results.

The string hash function is modified from a code fragment attributed to Chris Torek:

```
     /*
      *  SLPhash returns a hash value in the range 0-1023 for a
      *  string of single-byte characters, of specified length.
      */
     unsigned long SLPhash (const char *pc, unsigned int length)
         unsigned long h = 0;
 while (length-- != 0) {
             h *= 33;
             h += *pc++;
         }
         return (0x3FF & h);  /* round to a range of 0-1023 */
     }
```

This value is added to the base range of Service Specific Discovery Addresses, to be assigned by IANA. These will be 1024 contiguous multicast addresses.

3.7. Service Location Scaling, and Multicast Operating Modes

In a very small network, with few nodes, no DA is required. A user agent can detect services by multicasting requests. Service Agents will then reply to them. Further, Service Agents which respond to user requests must be used to make service information available. This does not scale to environments with many hosts and services.

When scaling Service Location systems to intermediate sized networks, a central repository (Directory Agent) may be added to reduce the number of Service Location messages transmitted in the network infrastructure. Since the central repository can respond to all Service and Attribute Requests, fewer Service and Attribute Replies will be needed; for the same reason, there is no need to differentiate between Directory Agents.

A site may also grow to such a size that it is not feasible to maintain only one central repository of service information. In this case more Directory Agents are needed. The services (and service agents) advertised by the several Directory Agents are collected together into logical groupings called "Scopes".

All Service Registrations that have a scope must be registered with all DAs (within the appropriate multicast radius) of that scope which have been or are subsequently discovered. Service Registrations which have no scope are only registered with unscoped DAs. User Agents make requests of DAs whose scope they are configured to use.

Service Agents MUST register with unscoped DAs even if they are configured to specifically register with DAs which have a specific scope or set of scopes. User Agents MAY query DAs without scopes, even if they are configured to use DAs with a certain scope. This is because any DA with no scope will have all the available service information.

Scoped user agents SHOULD always use a DA which supports their configured scope when possible instead of an unscoped DA. This will prevent the unscoped DAs from becoming overused and thus a scaling problem.

It is possible to specially configure Service Agents to register only with a specific set of DAs (see Section 22.1). In that case, services may not be available to User Agents via all Directory Agents, but some network administrators may deem this appropriate.

There are thus 3 distinct operating modes. The first requires no administrative intervention. The second requires only that a DA be run. The last requires that all DAs be configured to have scope and that a coherent strategy of assigning scopes to services be followed. Users must be instructed which scopes are appropriate for them to use. This administrative effort will allow users and applications to subsequently dynamically discover services without assistance.

The first mode (no DAs) is intended for a LAN. The second mode (using a DA or DAs, but not using scopes) scales well to a group of interconnected LANs with a limited number of hosts. The third mode (with DAs and scopes) allows the SLP protocol to be used in an internetworked campus environment.

If scoped DAs are used, they will not accept unscoped registrations or requests. UAs which issue unscoped requests will discover only unscoped services. They SHOULD use a scope in their requests if possible and SHOULD use a DA with their scope in preference to an unscoped DA. In a large campus environment it would be a bad idea to have ANY unscoped DAs: They attract ALL registrations and will thus present a scaling problem eventually.

A subsequent protocol document will describe mechanisms for supporting a service discovery protocol for the global Internet.

4. Service Location General Message Format

The following header is used in all of the message descriptions below and is abbreviated by using "Service Location header =" followed by the function being used.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    Version    |    Function   |           Length              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|O|M|U|A|F| rsvd|    Dialect    |        Language Code          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|        Char Encoding          |              XID              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Version This protocol document defines version 1 of the Service Location protocol.

Function Service Location datagrams can be identified as to their operation by the function field. The following are the defined operations:

Message Type	Abbreviation	Function Value
Service Request	SrvReq	1
Service Reply	SrvRply	2
Service Registration	SrvReg	3
Service Deregister	SrvDereg	4
Service Acknowledge	SrvAck	5
Attribute Request	AttrRqst	6
Attribute Reply	AttrRply	7
DA Advertisement	DAAdvert	8
Service Type Request	SrvTypeRqst	9
Service Type Reply	SrvTypeRply	10

Length   The number of bytes in the message, including the Service Location Header.

O   The 'Overflow' bit. See Section 18 for the use of this field.

M	The ’Monolingual’ bit. Requests with this bit set indicate the User Agent will only accept responses in the language (see section 17) that is indicated by the Service or Attribute Request.
U	The ’URL Authentication Present’ bit. See sections 4.2, 4.3, 9, and 11 for the use of this field.
A	The ’Attribute Authentication Present’ bit. See sections 4.2, 4.3, and 13 for the use of this field.
F	If the ’F’ bit is set in a Service Acknowledgement, the directory agent has registered the service as a new entry, not as an updated entry.
rsvd	MUST be zero.
Dialect	Dialect tags will be used by future versions of the Service Location Protocol to indicate a variant of vocabulary used. This field is reserved and MUST be set to 0 for compatibility with future versions of the Service Location Protocol.

Language Code
Strings within the remainder of the message which follows are to be interpreted in the language encoded (see section 17 and appendix A) in this field.

Character Encoding
The characters making up strings within the remainder of the message may be encoded in any standardized encoding (see section 17.1).

Transaction Identifier (XID)
The XID (transaction ID) field allows the requester to match replies to individual requests (see section 4.1).

Note that, whenever there is an Attribute Authentication block, there will also be a URL Authentication block. Thus, it is an error to have the ’A’ bit set without also having the ’U’ bit set.

4.1. Use of Transaction IDs (XIDs)

Retransmission is used to ensure reliable transactions in the Service Location Protocol. If a User Agent or Service Agent sends a message and fails to receive an expected response, the message will be sent again. Retransmission of the same Service Location datagram should not contain an updated XID. It is quite possible the original request reached the DA or SA, but reply failed to reach the requester. Using the same XID allows the DA or SA to cache its reply to the original request and then send it again, should a duplicate request arrive. This cached information should only be held very briefly (CONFIG_INTERVAL_0.) Any registration or deregistration at a Directory Agent, or change of service information at a SA should flush this cache so that the information returned to the client is always valid.

The requester creates the XID from an initial random seed and increments it by one for each request it makes. The XIDs will eventually wrap back to zero and continue incrementing from there.

Directory Agents use XID values in their DA Advertisements to indicate their state (see section 15.2).

4.2. URL Entries

When URLs are registered, they have lifetimes and lengths, and may be authenticated. These values are associated with the URL for the duration of the registration. The association is known as a "URL-entry", and has the following format:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Lifetime            |        Length of URL          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
\                              URL                              \
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|              (if present) URL Authentication Block .....
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

```
 Lifetime   The length of time that the registration is valid, in
          the absence of later registrations or deregistration.

 Length of URL
          The length of the URL, measured in bytes and < 32768.

 URL Authentication Block
          (if present) A timestamped authenticator (section 4.3)
```

The URL conforms to RFC 1738 [6]. If the 'U' bit is set in the message header, the URL is followed by an URL Authentication Block. If the scheme used in the URL does not have a standardized representation, the minimal requirement is:

```
service:<srvtype>://<addr-spec>
```

"service" is the URL scheme of all Service Location Information included in service registrations and service replies. Each URL entry contains the service:<srvtype> scheme name. It may also include an <addr-spec> except in the case of a reply to a Service Type request (see section 7).

4.3. Authentication Blocks

Authentication blocks are used to authenticate service registrations and deregistrations. URLs are registered along with an URL Authentication block to retain the authentication information in the URL entry for subsequent use by User Agents who receive a Service Reply containing the URL entry. Service attributes are registered along with an Attribute Authentication block. Both authentication blocks have the format illustrated below.

If a service registration is accompanied by authentication which can be validated by the DA, the DA MUST validate any subsequent service deregistrations, so that unauthorized entities cannot invalidate such registered services. Likewise, if a service registration is accompanied by an Attribute Authentication block which can be validated by the DA, the DA MUST validate any subsequent attribute registrations, so that unauthorized entities cannot invalidate such registered attributes.

To avoid replay attacks which use previously validated deregistrations, the deregistration or attribute registration message must contain a timestamp for use by the DA. To avoid replay attacks which use previously validated registrations to nullify a valid deregistration, registrations must also contain a timestamp.

An authentication block has the following format:

```
    0                   1                   2                   3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                                                               |
   +                           Timestamp                           +
   |                                                               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |  Block Structure Descriptor   |            Length             |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |            Structured Authenticator ...
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Timestamp A 64-bit value formatted as specified by the Network Time Protocol (NTP) [16].

Block Structure Descriptor (BSD)
A value describing the structure of the Authenticator. The only value currently defined is 1, for Object-Identifier.

Length The length of the Authenticator

Structured Authenticator
An algorithm specification, and the authentication data produced by the algorithm.

The Structured Authenticator contains a digital signature of the information being authenticated. It contains sufficient information to determine the algorithm to be used and the keys to be selected to verify the digital signature.

The digital signature is computed over the following ordered stream of data:

```
    CHARACTER ENCODING OF URL   (2 bytes in network byte order)
    LIFETIME                    (2 bytes in network byte order)
    LENGTH OF URL               (2 bytes in network byte order)
    URL                         (n bytes)
    TIMESTAMP                   (8 bytes in SNTP format [16])
```

When producing a URL Authentication block, the authentication data produced by the algorithm identified within the Structured Authenticator calculated over the following ordered stream of data:

```
    ATTRIBUTE CHARACTER ENCODING   (2 bytes in network byte order)
    LENGTH OF ATTRIBUTES           (2 bytes in network byte order)
    ATTRIBUTES                     (n bytes)
    TIMESTAMP                      (8 bytes in SNTP format [16])
```

Every Service Location Protocol entity (User Agent, Service Agent, or Directory Agent) which is configured for use with protected scopes SHOULD implement "md5WithRSAEncryption" [4] and be able to associate it with BSD value == 1.

In the case where BSD value == 1 and the OID "md5WithRSAEncryption" is selected, the Structured Authenticator will start with the ASN.1 Distinguished Encoding (DER) [9] for "md5WithRSAEncryption", which has the as its value the bytes (MSB first in hex):

```
   "30 0d 06 09 2a 86 48 86 f7 0d 01 01 04 05 00"
```

This is then immediately followed by an ASN.1 Distinguished Encoding (as a "Bitstring") of the RSA encryption (using the Scope’s private key) of a bitstring consisting of the OID for "MD5" concatenated by the MD5 [22] message digest computed over the fields above. The exact construction of the MD5 OID and digest can be found in RFC 1423 [4].

4.4. URL Entry Lifetime

The Lifetime field is set to the number of seconds the reply can be cached by any agent. A value of 0 means the information must not be cached. User Agents MAY cache service information, but if they do, they must provide a way for applications to flush this cached information and issue the request directly onto the network.

Services should be registered with DAs with a Lifetime, the suggested value being CONFIG_INTERVAL_1. The service must be reregistered before this interval elapses, or the service advertisement will no longer be available. Thus, services which vanish and fail to deregister eventually become automatically deregistered.

5. Service Request Message Format

The Service Request is used to obtain URLs from a Directory Agent or Service Agents.

APPENDIX X

HP Jornada 600 Series Handheld PC

User's Guide

HEWLETT® PACKARD

Part number F1813-90001
Printed in Singapore
Edition 1

Copyright notice

Hewlett-Packard Singapore (Pte) Ltd.
Asia Pacific Personal Computer Division
452 Alexandra Road
Singapore 119961

Welcome | 1

Congratulations on purchasing the Hewlett-Packard Jornada 600 Series Handheld PC (H/PC), a mobile device powered by the Microsoft® Windows® CE operating system. The HP Jornada 600 Series H/PC is available in four models: the HP Jornada 680e, 680, 690e, and 690. This User's Guide includes instructions for all four models. The differences in specifications are:

- Model 680e—no built-in modem, 16MB RAM.
- Model 680—built-in modem, 16MB RAM.
- Model 690e—no built-in modem, 32MB RAM.
- Model 690—built-in modem, 32MB RAM.
  (Note that on models 690 and 690e, a maximum of 16MB can be allocated for storage memory.)

If you are familiar with Microsoft Windows products and notebook PCs, you will notice that your HP Jornada has many of the same characteristics, making it easy for you to be productive quickly.

In this chapter, you will find:

- An overview of this User's Guide and other sources of help and information
- A list and brief descriptions of the built-in applications, including Microsoft Windows CE, Handheld PC Professional Edition software and special HP applications
- An introduction to HP Jornada features and a description of the advantages of using the HP Jornada

Using this guide

This User's Guide will give you a quick and effective introduction to your HP Jornada. Although great care has been taken to ensure the accuracy of procedures and artwork, some of the screens displayed on your HP Jornada may differ from the ones that appear in this User's Guide.

Detailed, step-by-step instructions for using the programs on your HP Jornada are also included in online Help, so you do not have to carry this guide when you travel. (For more information on using online Help, see the **Using Help on your HP Jornada** section in chapter 2.)

Conventions

This User's Guide includes certain visual cues that will help you find the information you want easily.

💡	A shortcut, another way to do something, or expanded information about the topic.
❗	Caution or warning information about the topic. This is important information that should be observed to prevent loss of data or damage to your HP Jornada.
📝	Helpful information related to the topic.

Finding information

This guide describes what is included with your HP Jornada, provides an overview of the installed programs, and describes how to set up communications between your device and desktop PC and between your device and the Internet. The following table lists the different types of information available to help you use your HP Jornada. Note that although this book introduces the programs on your device, it does not describe them completely. For more information, see the comprehensive online Help for each program.

Information	**Source**
Programs	This User's Guide –or– Online Help on your device. On the **Start** menu, tap **Help**.
Unfamiliar technical terms	The glossary located at the end of this User's Guide.
Synchronizing and exchanging files with a desktop PC (including information on connecting while travelling	This User's Guide –or– Online Help on your device. On the **Start** menu, tap **Help**. –or– Windows CE Services online Help on your desktop PC. In the Mobile Devices window, click **Help**, and then click **Windows CE Services Help Topics**.
ToolTips	Any toolbar button. Tap and hold the button and the button name will appear. (To avoid activating the toolbar button, drag off the button before lifting the stylus.) For more information, see the **Using your HP Jornada** section in chapter 2.
Troubleshooting information on connections	This User's Guide –or– The Communications Troubleshooter on your desktop PC. In the Mobile Devices window, click **Help**, and then click **Communications Troubleshooter**.
Software updates, device drivers, and lists of com-patible accessories	The Hewlett-Packard Jornada Web site at www.hp.com/jornada.
Up-to-date information on Windows CE–based devices	The Microsoft Windows CE Web site at www.microsoft.com/windowsce.
Information on this release of Windows CE	Readme.doc file (located in the Windows CE Services folder on your desktop PC).

The complete text of this User's Guide is available on the HP Web site at www.hp.com/jornada. You can download the User's Guide to your desktop PC and view it using the Adobe™ Acrobat Reader, available from the Adobe Web site at www.adobe.com.

Learning about HP Jornada

Hewlett-Packard has taken mobile computing to new levels of compatibility and convenience. With HP Jornada, you'll notice many benefits not found in other, similar computers. For example you can:

- Move the data you already have on an older palmtop or handheld PC to your HP Jornada. You can transfer information from older Windows CE devices and even some non–Windows CE devices. (See the **Transferring PIM data from older palmtop PCs** section in chapter 3.)
- Use e-mail without a modem. Conveniently send e-mail automatically when you synchronize with your desktop PC. (See the **Synchronizing data** section in chapter 3.)
- Take more data with you. A desktop Microsoft Word file may take up to 85 percent less space on your HP Jornada. (See the **Transferring files** section in chapter 3.)
- Send voice messages to friends and colleagues—even those who do not have Windows CE devices. You can save voice messages in the compatible Wave audio format. (See the **Recording voice memos** section in chapter 4.)
- Keep Word, Access, and Excel files stored at work and at home synchronized. Update any file in any location, and once you synchronize, the file will be updated in every place it is stored. (See the **Synchronizing data** section in chapter 3.)
- Read online news and information while offline, or download Web pages for viewing later. (See the **Browsing the Web** section in Chapter 6.)
- Work any time, any place, with up to 7 hours of battery life. (See the **Managing battery power** section in chapter 3.)
- Synchronize mail messages, contacts, appointments, and tasks with your desktop or notebook PC in just a few steps. (See the **Synchronizing data** section in chapter 3.)
- Carry your HP Jornada with you. The HP Jornada weighs only 510 g (1.1 lb), including batteries.

- Expand your functionality with PC Card accessories. (See the **Accessories** section in chapter 7.)
- Get it all in one package. Your HP Jornada comes complete with a built-in modem (models 680 and 690 only).

HP Jornada is your mobile computing companion. Several valuable features allow you to stay organized and in touch while you're on the road. For instance:

- The HP Jornada viewer application displays PIM data (contacts, calendar, and tasks) at a glance, allowing you to navigate to or view the data you need immediately. You can even view notes attached to appointments.
- The HP Jornada dialup application leads you through the steps required to configure a dial-up connection and to access your e-mail and the Web. When you are ready to connect, just use the convenient dialup window on your HP Jornada desktop.
- The HP Jornada quick pad provides a convenient place to jot down notes, names, telephone numbers, and other data for short-term storage. Information can be saved in quick pad for fast retrieval or sent to the appropriate database for long-term storage.
- The HP Jornada backup application can back up and restore information to/from an optional CompactFlash Card or PC Card, safeguarding against loss of data while you are on the road—even if power is lost.

HP Jornada programs

Your HP Jornada already includes the full suite of software that you need to function as a mobile professional. Detailed information can be found in later chapters.

The programs listed below are pre-installed in ROM, so you will never need to reinstall them.

Microsoft Pocket Office

On the **Start** menu, point to **Programs**, point to **Office**, and then tap one of the following choices.

- **Pocket Word**—Take notes and compose documents, or transfer Microsoft Word files from your desktop PC to read and review while you are away from your office.

- **Pocket Excel**—View and edit price lists or financial forecasts on your HP Jornada, or fill out your expense form before you even land at the home office.
- **Pocket Access**—Take database information with you on the road, and fill in custom forms to update Access databases when you return to your office.
- **Pocket PowerPoint**—Create professional presentations on your desktop, and then take them with you to show on your HP Jornada. Or, use a PC Card VGA adapter (F1252A) to project them to an external monitor or VGA projector.

Microsoft Pocket Outlook

On the **Start** menu, point to **Programs**, point to **Pocket Outlook**, and then tap one of the following choices.

- **Calendar**—Never miss a meeting. Keep track of important dates and events or manage your schedule.
- **Contacts**—Take your address book with you so that you always have access to names, addresses, and telephone numbers. If you update your HP Jornada contacts list, just synchronize with your desktop PC and your contacts will always be up to date.
- **Inbox**—Send and receive e-mail messages and synchronize your HP Jornada Inbox with Microsoft Outlook™ or Exchange™ on your desktop PC partner in a matter of minutes.
- **Tasks**—Keep track of to-do lists. Set an alarm or a reminder and HP Jornada will make sure you do not forget a task!

Microsoft Explorers

On the **Start** menu, point to **Programs**, and then tap **Internet Explorer** or **Windows Explorer**. Or, double-tap the **Internet Explorer** or **My Handheld PC** icons on the desktop.

- **Pocket Internet Explorer**—Browse the Web from your HP Jornada or subscribe to channel content with this streamlined version of Microsoft Internet Explorer 3.0.
- **Windows Explorer**—Browse the files and folders on your HP Jornada.

Communication

On the **Start** menu, point to **Programs**, point to **Communication**, and then tap one of the following choices.

- **ActiveSync™**—Synchronize your HP Jornada with your desktop or notebook PC over a network or dial-up connection from a remote location.
- **PC Link**—Establish the connection between your HP Jornada and desktop PC partner with a single tap.
- **Remote Networking**—Connect to a dial-up server, RAS account, or Internet service provider.
- **Terminal**—Connect to online services that require VT-100 or TTY terminal emulation.
- **bFAX Pro**—Send Word documents or typed notes and receive faxes with bFAX Pro from bSquare.

Special HP Applications

On the **Start** menu, point to **Programs**, point to **HP Applications**, and then tap one of the following choices.

- **HP viewer**—Display Calendar, Task, and Contact information (Pocket Outlook) in as much detail as you need so that you are always in control of your schedule. Month View, Week View, and Day View help you manage your calendar even more efficiently. (HP viewer is also accessible through a desktop icon and an HP hot key.)
- **HP dialup**—Configure connections to the Internet and e-mail accounts or corporate network, and then dial in from a convenient pop-up window on your HP Jornada desktop. (HP dialup is also accessible through a desktop icon.)
- **HP quick pad**—Write simple notes and reminders on this electronic notepad, and then move the important information to Pocket Outlook or a Word document.
- **HP backup**—Protect your valuable data even when you are away from your office by backing up your Pocket Outlook data or your entire device to a CompactFlash or PC Card.

On the **Start** menu, point to **Programs**, point to **HP Utilities**, and then tap one of the following choices.

- **HP settings**—Adjust screen controls and sound volume to suit any work environment. Or, choose from four preset profiles for different environments, and change all options with the touch of a button.
- **HP hot keys**—Open programs, files, or folders with a single keystroke. The HP Jornada hot keys and hard icons are fully customizable, so you can configure them for one-touch access to your favorite programs or frequently used documents.
- **HP macro**—Automate common tasks using a powerful scripting language. HP Macro allows you to create a set of recorded commands and actions that you can repeat or play back by pressing a specific key combination.

Accessories

On the **Start** menu, point to **Accessories**, and then tap one of the following choices.

- **Microsoft InkWriter**®—Jot quick notes or create sketches while in a meeting.
- **Microsoft Voice Recorder**—Record voice memos or vital information.
- **Calculator**—Perform simple calculations in an on-screen calculator, and then copy the results to any open document.
- **World Clock**—Keep track of the time anywhere in the world and display useful travel information for both your home city and the city you are visiting.
- **bFIND**®—Search for a word or text string in any database, file, file name, or e-mail message stored on your HP Jornada.
- **OmniSolve**®—Perform complex mathematical and business calculations with this full-featured calculator from Landware.

Games

- **Solitaire**—Pass the time at the airport, on the train, or during a not-so-interesting meeting or lecture with this classic game.

Optimizing your HP Jornada | 7

This chapter details ways you can optimize your HP Jornada using Control Panel and HP Utilities, and ways you can expand the features and functionality of your HP Jornada by adding software and accessories. This chapter also offers tips on traveling with your HP Jornada and keeping your HP Jornada safe. These tips will help you become more efficient and make working with your HP Jornada more comfortable and fun.

In the following pages, you will learn about:

- Adding to your HP Jornada—Add programs, fonts, sounds, and desktop wallpaper.
- Using accessories—Use CompactFlash and PC Cards with the expansion module.
- Managing memory—Learn how to allocate storage and program memory for optimal performance.
- Managing battery power—Learn how to replace, and how to get the most from your batteries.
- Fine-tuning performance—Use control panel to set various options, including customizing the HP hot keys and setting general system settings.
- Traveling with your HP Jornada—Follow simple guidelines for optimizing your device for use while away from the office, including a check list of everything you need to take with you.
- Protecting your HP Jornada—Safeguard your HP Jornada from theft and data loss.
- Creating system macros—Automate common tasks with the HP macro utility.

Adding programs, fonts, sounds, and images

This section describes how to add functionality to your HP Jornada by installing software and how to customize the Windows CE desktop and working environment.

Installing programs

You can add even more functionality to your HP Jornada by installing third-party software. A wide variety of commercial software is available for the Microsoft Windows CE operating system, ranging from custom business applications and system utilities to games and entertainment. Some programs are available on the HP Jornada Handheld PC compact disc and the Desktop Software for Microsoft Windows CE compact disc included with your HP Jornada. Other programs are available from software distributors and on the Worldwide Web.

The only programs that will run on your HP Jornada are those designed specifically for Windows CE. You cannot run programs designed for Windows 95 or Windows NT on your HP Jornada.

The HP Jornada Handheld PC compact disc includes many productivity-enhancing programs such as:

- Inso® Outisde In® file viewer software for viewing popular desktop file formats
- Pocket Quicken™ from On The Go Software (for U.S. only)
- TrueSync® CE 2.0 from Starfish Software for synchronizing your HP Jornada with the REX™ Classic and REX Pro Cards
- Trio Phone Manager for sending/receiving SMS with a GSM phone (for Europe and Asia Pacific only)
- WestTek™ JetCet™ color printing utilities (30-day trial)
- Image Expert CE from Sierra Imaging
- Java for Windows CE from Microsoft Corporation
- HP PIM Translation Utility

The minimum system requirements for installing programs from the HP Jornada Handheld PC compact disc are as follows:

- Microsoft Windows 95, Windows 98, or Windows NT 4.0

- Desktop PC with a 486/66 or higher processor
- 2X or faster CD-ROM drive
- VGA or higher resolution graphics card
- Web browser (Netscape Navigator 2.0 or higher or Microsoft Internet Explorer 2.0 or higher)

Explore the HP Jornada Handheld PC compact disc to enjoy these free software programs. For more information about a particular program, go to the Web site of the manufacturer of that program.

Many other useful programs are available from the Hewlett-Packard Web site at www.hp.com/jornada. Hewlett-Packard does not support the use of programs that have not been certified by Microsoft.

Typically, you install software to your H/PC by first loading the installation files onto your desktop PC, as described in the following procedure.

If a program is designed for direct installation, you may be able to download or install the program from the Web to your HP Jornada. The Web site should provide instructions for installing the program.

To install software

1. Download the software program or installer from the Web to your desktop PC.
   –or–
   Insert the floppy disk or compact disc into the appropriate drive on your desktop PC.
2. Connect your HP Jornada to your desktop PC using the docking cradle or sync cable and establish a partnership. (For more information, see chapter 4.).
3. If the program includes an installer program (typically named Setup.exe or Install.exe), double-click the installer program on the desktop PC. The installer program will copy the necessary files to your HP Jornada.
   –or–
   If the program does not have an associated installer or setup program, drag the program file (typically an *.exe file type) to the HP Jornada icon in the Mobile Devices window. If the **No converter selected** dialog box appears, click **OK** to copy the file without conversion.

4. If prompted by the installer, perform a soft reset of your H/PC. For more information, see the **Resetting** section in chapter 8.

Install software by dragging the program file to the Mobile Devices window only if no installer program is available. Software installed in this way may not appear on the **Remove Programs** list, and you may have to manually delete the program if you wish to remove it from your HP Jornada.

After you have installed a program on your HP Jornada, you can use Windows CE Application Manager to remove the program or to reinstall the program after it has been removed. If you do not have enough storage memory on your HP Jornada, you may want to use Application Manager to temporarily remove programs you no longer use or programs that you use infrequently.

To add or remove programs with Application Manager

1. Connect your HP Jornada to your desktop PC, and then open the Mobile Devices window.
2. On the **Tools** menu in the Mobile Devices window, click **Application Manager**.
3. In the list of programs, select the program you wish to install, and then click **Add**.
–or–
Select the program you want to delete, and then click **Remove**.

Removing programs

You can use either Application Manager (on your desktop PC) or the Remove Programs control panel (on your HP Jornada) to remove programs.

To remove a program from your HP Jornada with the Remove Programs control panel

1. On the **Start** menu, point to **Settings**, and then tap **Control Panel**.
2. Double-tap the Remove Programs icon.
3. In the **Programs** list, select the program you want to remove.
4. Tap **Remove**.

Programs stored in ROM cannot be removed. (For a list of these programs see the **HP Jornada programs** section in chapter 1.)