PROVISIONING GUIDE

Cisco Small Business

Voice System, Voice Gateways, and IP Telephones

OL-19687-01

Contents

About This Document

This guide describes the provisioning of Cisco Small Business Voice over IP (VoIP) products. It contains the following sections:

Purpose

The following Cisco Small Business VoIP products can be remotely provisioned or preprovisioned using the information in this document:

- SPA9000—IP PBX with Auto-Attendant; can be used with the SPA400, which provides a SIP-PSTN gateway
- Cisco Small Business Analog Telephone Adapters (ATAs):
  - PAP2T—Voice adapter with two FXS ports
  - SPA2102—Voice adapter with router
  - SPA3102—Voice adapter with router and PSTN connectivity
  - SPA8000—Voice adapter supporting up to eight FXS connections
  - WRP400—Wireless-G ADSL gateway with two FXS ports
- Cisco Small Business IP phones:
  - SPA901—One line, small, affordable, no display
  - SPA921—One-line business phone
  - SPA922—One-line business phone with Power over Ethernet (PoE) support and an extra 10/100 Ethernet port for connecting another device to the LAN

- SPA941—Four-line business phone.
- SPA942—Four-line business phone. Power over Ethernet (PoE) support and an extra 10/100 Ethernet port for connecting another device to the LAN
- SPA962—Six lines, hi-resolution color display. Power over Ethernet (PoE) support and an extra 10/100 Ethernet port for connecting another device to the LAN
- SPA525G--Five lines, hi-resolution color display. Power over Ethernet (PoE), 10/100 switch, BlueTooth, WiFi 802.11g, USB port, MP3 player.
- WIP310—One line, hi-resolution color display. WiFi 802.11g

Document Audience

This document is written for service providers who offer services using Cisco Small Business VoIP products and specifically for administrative staff responsible for remote provisioning and preprovisioning Cisco Small Business devices.

Organization

This document is divided into the following chapters and appendices.

Chapter	Contents
Chapter 1, "Provisioning Cisco Small Business VoIP Devices"	This chapter introduces Cisco Small Business VoIP products.
Chapter 2, "Creating Provisioning Scripts"	This chapter describes how to work with Cisco Small Business provisioning scripts and configuration profiles.
Chapter 3, "Provisioning Tutorial"	This chapter provides step-by-step procedures for using the scripting language to create a configuration profile.
Chapter 4, "Provisioning Field Reference"	This chapter provides a systematic reference for each parameter on the Provisioning tab of the administration web server.

Chapter	Contents
Appendix A, “Example Configuration Profile”	This appendix contains a sample profile that you may find helpful.
Appendix B, “Acronyms”	This appendix provides the expansion of acronyms used in this document.
Appendix C, “Where to Go From Here”	This appendix provides links to resources for information and support.

Finding Information in PDF Files

The guides for Cisco Small Business products are available as PDF files. The PDF Find/Search tool within Adobe® Reader® lets you find information quickly and easily online. You can perform the following tasks:

- Search an individual PDF file.
- Search multiple PDF files at once (for example, all PDFs in a specific folder or disk drive).
- Perform advanced searches.

Finding Text in a PDF

Follow this procedure to find text in a PDF file.

STEP 1 Enter your search terms in the Find text box on the toolbar.

NOTE By default, the Find tool is available at the right end of the Acrobat toolbar. If the Find tool does not appear, choose **Edit > Find**.

STEP 2 Optionally, click the arrow next to the Find text box to refine your search by choosing special options such as Whole Words Only.

STEP 3 Press **Enter**.

STEP 4 Acrobat displays the first instance of the search term.

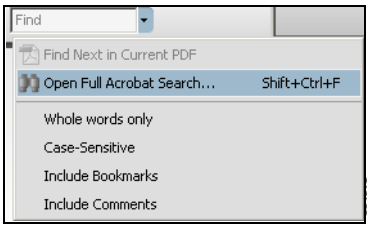
STEP 5 Press **Enter** again to continue to more instances of the term.

Finding Text in Multiple PDF Files

The *Search* window lets you search for terms in multiple PDF files that are stored on your PC or local network. The PDF files do not need to be open.

STEP 1 Start Acrobat Professional or Adobe Reader.

STEP 2 Choose **Edit** > **Search**, or click the arrow next to the *Find* box and then choose **Open Full Acrobat Search**.



STEP 3 In the *Search* window, complete the following steps:

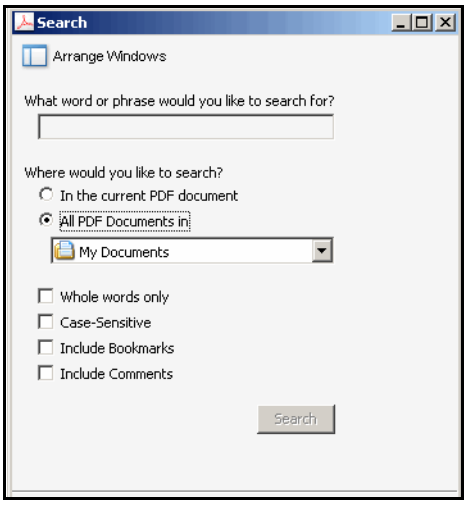
a. Enter the text that you want to find.

b. Choose **All PDF Documents in**.

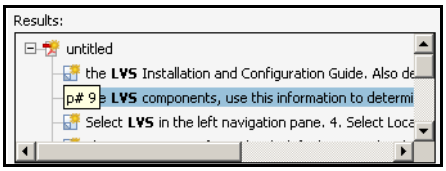
From the drop-down box, choose **Browse for Location**. Then choose the location on your computer or local network, and click **OK**.

c. If you want to specify additional search criteria, click **Use Advanced Search Options**, and choose the options you want.

d. Click **Search**.



STEP 4 When the Results appear, click + to open a folder, and then click any link to open the file where the search terms appear.



For more information about the Find and Search functions, see the Adobe Acrobat online help.

Document Conventions

The following typographic conventions are used in this document.

Typographic Element	Meaning
Boldface	Indicates an option on a menu or a literal value to be entered in a field.
<parameter>	Angle brackets (<>) are used to identify parameters that appear on the configuration pages of the administration web server. The index at the end of this document contains an alphabetical listing of each parameter, hyperlinked to the appropriate table in **Chapter 4, "Provisioning Field Reference"**
Italic	Indicates a variable that should be replaced with a literal value.
Monospaced Font	Indicates code samples or system output.

1

Provisioning Cisco Small Business VoIP Devices

This chapter describes the features and functionality available when provisioning Cisco Small Business IP Telephony Devices and explains the setup required. It includes the following sections:

Residential Deployment Provisioning Requirements

Cisco Small Business IP Telephony Devices are primarily intended for high-volume deployments by VoIP service providers to residential and small business customers. These devices are likely to be widely distributed across the Internet, connected through routers and firewalls at the customer premises. Further, IP Telephony Devices may serve as terminal nodes in business or enterprise environments, where the units may be operated within a self-contained LAN environment.

The IP Telephony Device can be seen as a remote extension of the service provider back-end equipment. Remote management and configuration is required to efficiently ensure proper operation of the IP Telephony Device at the customer premises.

Device configuration varies according to the individual customer and with the same customer over a period of time. The IP Telephony Device must be configured to match the account service parameters for the individual customer. Also, the configuration may need to be modified because of new service provider features, modifications in the service provider network, or firmware upgrades in the endpoint.

This customized, ongoing configuration is supported by the following features:

- Reliable remote control of the endpoint
- Encryption of the communication controlling the endpoint
- Streamlined endpoint account binding

Remote Endpoint Control

The service provider must be able to modify the configuration parameters in the IP Telephony Device after the unit has been deployed to the customer premises. The service provider must also be able to upgrade the firmware remotely, and both of these operations must be reliable.

In a residential deployment, the end IP Telephony Device is typically connected to a local network. The device accesses the Internet through a router using network address translation (NAT). For enhanced security, the router may attempt to block unauthorized incoming packets by implementing symmetric NAT, a packet filtering strategy which severely restricts the packets that are allowed to enter the protected network from the Internet.

Communication Encryption

The configuration parameters communicated to the IP Telephony Device may contain authorization codes or other information that need to be protected from unauthorized access. It is in the service provider's interest to prevent unauthorized activity by the customer, and in the customer's interest to prevent from unauthorized use of the account by other persons. For this reason, the service provider may wish to encrypt the configuration profile communication between the provisioning server and the IP Telephony Device, in addition to restricting access to the administration web server for the device.

Provisioning Overview

The Cisco Small Business IP Telephony Devices support secure remote provisioning and firmware upgrades. Configuration profiles can be generated by by using common, open source tools that facilitate integration into service provider provisioning systems. Supported transport protocols include TFTP, HTTP, and HTTPS with a client certificate. Cisco Small Business provisioning solutions are designed for high-volume residential deployment, where each IP Telephony Device typically resides in a separate LAN environment that is connected to the Internet with a NAT device.

An IP Telephony Device can be configured to resynchronize its internal configuration state to a remote profile periodically and on power up. A 256-bit symmetric key encryption of profiles is supported. In addition, an unprovisioned IP Telephony Device can receive an encrypted profile specifically targeted for that device without requiring an explicit key. Secure first-time provisioning is provided through a mechanism that uses SSL functionality.

NOTE Remote customization (RC) units are customized by Cisco so that when the unit is started, it tries to contact the Cisco provisioning server to download its customized profile.

User intervention is not required to initiate or complete a profile update or firmware upgrade. Remote firmware upgrade is achieved via TFTP or HTTP, but not using HTTPS because the firmware does not contain sensitive information that can be read by a customer. The upgrade logic is capable of automating multi-stage upgrades, if intermediate upgrades are required to reach a future upgrade state from an older release. A profile resync is only attempted when the IP Telephony Device is idle, because this may trigger a software reboot.

General purpose parameters are provided to help service providers to manage the provisioning process. Each IP Telephony Device can be configured to periodically contact a normal provisioning server (NPS). Communication with the NPS does not require the use of a secure protocol because the updated profile is encrypted by a shared secret key. The NPS can be a standard TFTP, HTTP or HTTPS server.

Initial Provisioning

Cisco Small Business IP Telephony Devices provide convenient mechanisms for initial provisioning, based on two deployment models:

- Retail distribution

  In this model, the customer purchases the IP Telephony Device from a retail outlet and subsequently requests VoIP service from the service provider. The service provider must then support secure remote configuration of the unit.

- Bulk distribution

  In this model, the service provider issues the IP Telephony Device to the customer as part of the VoIP service contract (RC units). The service provider acquires IP Telephony Devices in bulk quantity, and either preprovisions the IP Telephony Devices in-house or purchases RC units from Cisco.

Deploying RC Units

The in-house preprovisioning step can be eliminated by using RC units. Customization of RC units reduces the need to handle the units prior to shipping to end customers. It also discourages the use of Cisco Small Business IP Telephony Devices with a different service provider.

In this scenario, the MAC address of each RC unit is associated with a customized profile on a provisioning server that is maintained by Cisco for the Service Provider that purchased the units. The RC unit is preprovisioned by Cisco with the connection information for the Cisco Small Business provisioning server. When the RC unit is started, it tries to contact the Cisco Small Business provisioning server and download its customized profile.

The status of customization for an RC unit can be determined by using the administration web server and viewing the Info tab > Product Information page, Customization section. An RC unit that has not been provisioned displays Pending. An RC unit that has been provisioned displays the name of the company that owns the unit. If the unit is not an RC unit the web page displays Open.

Cisco Small Business offers RC units to service providers for volume deployments of endpoints. Through customization, the manufacturing default values of a select number of parameters can be customized to meet the needs of individual service providers.

The following is a sample template for an RC unit:

```
Restricted Access Domains "domain.com, domain1.com, domain2.com";
Primary_DNS               * "x.y.w.z";
Secondary_DNS             * "a.b.c.d";
Provision_Enable          * "Yes";
Resync_Periodic           * "30";
Resync_Error_Retry_Delay  * "30";
Profile_Rule  * "http://prov.domain.com/sipura/profile?id=$MA";
```

The Restricted Access Domain parameter is configured with the actual domain names of up to a maximum of five domains. The Primary_DNS and Secondary_DNS parameters are configured with the actual domain names or IP addresses of the DNS servers available to the RC unit.

Redundant Provisioning Servers

The provisioning server may be specified as an IP address or as a fully qualified domain name (FQDN). The use of a FQDN facilitates the deployment of redundant provisioning servers. When the provisioning server is identified through a FQDN, the IP Telephony Device attempts to resolve the FQDN to an IP address through DNS. Only DNS A-records are supported for provisioning; DNS SRV address resolution is not available for provisioning. The IP Telephony Device continues to process A-records until the first server responds. If no server associated with the A-records responds, the IP Telephony Device logs an error to the syslog server.

Retail Provisioning

The firmware for each IP Telephony Device includes an administration web server that displays the internal configuration and accepts new configuration parameter values. The server also accepts a special URL command syntax for performing remote profile resync and firmware upgrade operations.

In a retail distribution model, a customer purchases a Cisco Small Business voice endpoint device, and subsequently subscribes to a particular service. The customer first signs on to the service and establishes a VoIP account, possibly through an online portal with an Internet Telephony Service Provider (ITSP). Subsequently, the customer binds the particular device to the assigned service account.

To do so, the unprovisioned IP Telephony Device is instructed to resync with a specific provisioning server through a resync URL command. The URL command typically includes an account PIN number or alphanumeric code to associate the device with the new account.

```
http://192.168.1.102/admin/resync?https://prov.supervoip.com/cisco-init/
1234abcd
```

In this example, a device at the DHCP-assigned IP address 192.168.1.102 is instructed to provision itself to the SuperVoIP service at prov.supervoip.com. The PIN number for the new account is 1234abcd. The remote provisioning server is configured to associate the IP Telephony Device that is performing the resync request with the new account, based on the URL and the supplied PIN. Through this initial resync operation, the IP Telephony Device is configured in a single step, and is automatically directed to resync thereafter to a permanent URL on the server. For example:

```
https://prov.supervoip.com/cisco
```

For both initial and permanent access, the provisioning server relies on the client certificate for authentication and supplies correct configuration parameter values based on the associated service account.

Automatic In-House Preprovisioning

Using the administration web server and issuing a resync URL is convenient for a customer in the retail deployment model, but it is not as convenient for preprovisioning a large number of units. In this case, you can use automatic in-house preprovisioning.

With the factory default configuration, an IP Telephony Device automatically tries to resync to a specific file on a TFTP server, whose IP address is offered as one of the DHCP-provided parameters. A service provider can connect each new IP Telephony Device to a LAN environment that is configured for preprovisioning. Any new IP Telephony Device connected to this LAN automatically resyncs to the local TFTP server, initializing its internal state in preparation for deployment. This preprovisioning step configures the URL of the provisioning server, among other parameters.

Subsequently, when a new customer signs up for service, the preprovisioned device can be simply bar-code scanned, to record its MAC address or serial number, before being shipped to the customer. Upon receiving the unit, the customer connects the unit to the broadband link, possibly through a router. On power-up the IP Telephony Device already knows the server to contact for its periodic resync update.

Configuration Access Control

Besides configuration parameters that control resync and upgrade behavior, the IP Telephony Device provides mechanisms for restricting end-user access to various parameters.

The firmware provides specific privileges for login to a User account and an Admin account. Both can be independently password protected.

- Admin Account: Allows the service provider to configure the device. The Admin account has full access to all IVR functions and to all administration web server parameters.
- User Account: Allows the user of the device to access basic interactive voice response (IVR) functions and to configure a subset of the administration web server parameters.

The service provider can restrict the user account in the following ways:

- The service provider can choose which configuration parameters are available to the User account.
- The service provider can completely disable any user access to the administration web server.
- The factory reset control using the IVR can be disabled via provisioning.
- The Internet domains accessed by the device for resync, upgrades, and SIP registration for Line 1 can be restricted.

Configuration Profiles

The configuration profile defines the parameter values for a specific IP Telephony Device. The configuration profile can be used in two formats:

- Open (XML-style) format

  The XML-style format lets you use standard tools to compile the parameters and values. To protect confidential information in the configuration profile, this type of file is generally delivered from the provisioning server to the IP Telephony Device over a secure channel provided by HTTPS.

- Proprietary, plain-text format

  The plain-text configuration file uses a proprietary format, which can be encrypted to prevent unauthorized use of confidential information. By convention, the profile is named with the extension .cfg (for example, spa962.cfg). The SIP Profiler Compiler (SPC) tool is provided for compiling the plain-text file containing parameter-value pairs into an encrypted CFG file. The SPC tool is available from Cisco for the Win32 environment (spc.exe) and Linux-i386-elf environment (spc-linux-i386-static). Availability of the SPC tool for the OpenBSD environment is available on a case-by-case basis. For more information, see **Downloading the SIP Profile Compiler (SPC) Tool, page 17**.

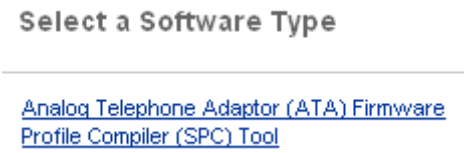
Downloading the SIP Profile Compiler (SPC) Tool

STEP 1 Go to Cisco.com, enter the model number in the search box, and then click **Go**.

STEP 2 In the **Filter Results By** list on the left side of the Search Results page, find **Task**, and then choose **Download Software**.

STEP 3 Click the **Download Software** link, which is usually the first link in the filtered list.

STEP 4 When the Select Software Type page appears, choose **Profile Compiler (SPC) Tool**.



STEP 5 In the next step, choose the latest release of firmware.

STEP 6 Follow the instructions on the screen to continue through the steps in the download process.

Provisioning States

The provisioning process involves four provisioning states, as described in the following table.

Flow Step	Step Description
MFG-RESET	Manufacturing Reset: The device returns to a fully unprovisioned state. All configurable parameters regain their manufacturing default values. Manufacturing reset can be performed through the following IVR sequence: ****RESET#1# Allowing the end user to perform manufacturing reset guarantees that the device can always be returned to an accessible state.
SP-CUST	Service Provider Customization: The Profile_Rule parameter is configured to point to a device-specific configuration profile, using a provisioning server that is specific to the service provider. There are three methods: ▪ Auto-configuration via local DHCP server. A TFTP server name or IPv4 address is specified by DHCP on the local network. The indicated TFTP server carries the desired Profile_Rule entry in the CFG file /spa962.cfg ▪ Enter a resync URL. A URL starts a web browser and requests a resync to a specific TFTP server by entering the following URL syntax: http://x.x.x.x/admin/resync?prvserv/*device.cfg* where x.x.x.x is the IP address of the IP Telephony Device and prvserv is the target TFTP server, and device.cfg is the name of the configuration file on the server. ▪ Edit Profile_Rule parameter. Open the provisioning pane on the web interface, and enter the TFTP URL in the Profile_Rule parameter: for example, prserv/spa962.cfg. ▪ The spa962.cfg file modifies the Profile_Rule to contact a specific TFTP server and to request a MAC-address specific CFG file. For example, the following entry contacts a specific provisioning server, requesting a new profile unique to this unit: `Profile_Rule tftp.callme.com/profile/$MA/spa962.cfg;`

Flow Step	Step Description

SEC-PRV-1	Secure Provisioning—Initial Configuration: The initial device-unique CFG file should be targeted to each IP Telephony Device by compiling the CFG file with the spc --target option. This provides an initial level of encryption that does not require the exchange of keys.
	
	The initial device-unique CFG file should reconfigure the profile parameters to enable stronger encryption, by programming a 256-bit encryption key, and pointing to a randomly generated TFTP directory. For example, the CFG file might contain:
	
	`Profile_Rule [--key $A] tftp.callme.com/profile/$B/spa962.cfg;`
	`GPP_A 8e4ca259…;  # 256 bit key`
	`GPP_B Gp3sqLn…;  # random CFG file path directory`

SEC-PRV-2	Secure Provisioning—Full Configuration: The subsequent profile resync operations retrieve 256-bit encrypted CFG files, which maintain the IP Telephony Device in a state synchronized to the provisioning server. All remaining parameters are configured and maintained through this strongly encrypted profile. The encryption key and random directory location can be changed periodically for extra security.

Using HTTPS

The IP Telephony Device provides a reliable and secure provisioning strategy based on HTTPS requests from the device to the provisioning server. Both a server certificate and a client certificate a used to authenticate the IP Telephony Device to the server and the server to the IP Telephony Device.

To use HTTPS, you must generate a Certificate Signing Request (CSR) and submit it to Cisco. Cisco generates a certificate for installation on the provisioning server. The IP Telephony Device accepts the certificate when it seeks to establish an HTTPS connection with the provisioning server. This procedure is described in the **“Enabling HTTPS” section on page 26**.

How HTTPS Works

HTTPS encrypts the communication between a client and a server, protecting the message contents from other intervening network devices. The encryption method for the body of the communication between a client and a server is based on symmetric key cryptography. With symmetric key cryptography, a single secret key is shared by a client and a server over a secure channel protected by Public/Private key encryption.

Messages encrypted by the secret key can only be decrypted using the same key. HTTPS supports a wide range of symmetric encryption algorithms. The IP Telephony Device implements up to 256-bit symmetric encryption, using the American Encryption Standard (AES), in addition to 128-bit RC4.

HTTPS also provides for the authentication of a server and a client engaged in a secure transaction. This feature ensures that a provisioning server and an individual client cannot be spoofed by other devices on the network. This is an essential capability in the context of remote endpoint provisioning.

Server and client authentication is performed by using public/private key encryption with a certificate that contains the public key. Text that is encrypted with a public key can be decrypted only by its corresponding private key (and vice versa). The IP Telephony Device supports the RSA algorithm for public/private key cryptography.

Server Certificate

Each secure provisioning server is issued an SSL server certificate, directly signed by Cisco. The firmware running on the IP Telephony Device recognizes only a Cisco certificate as valid. When a client connects to a server via HTTPS, it rejects any server certificate that is not signed by Cisco.

This mechanism protects the service provider from unauthorized access to the IP Telephony Device, or any attempt to spoof the provisioning server. Without such protection, an attacker might be able to reprovision the IP Telephony Device, to gain configuration information, or to use a different VoIP service.

Client Certificates

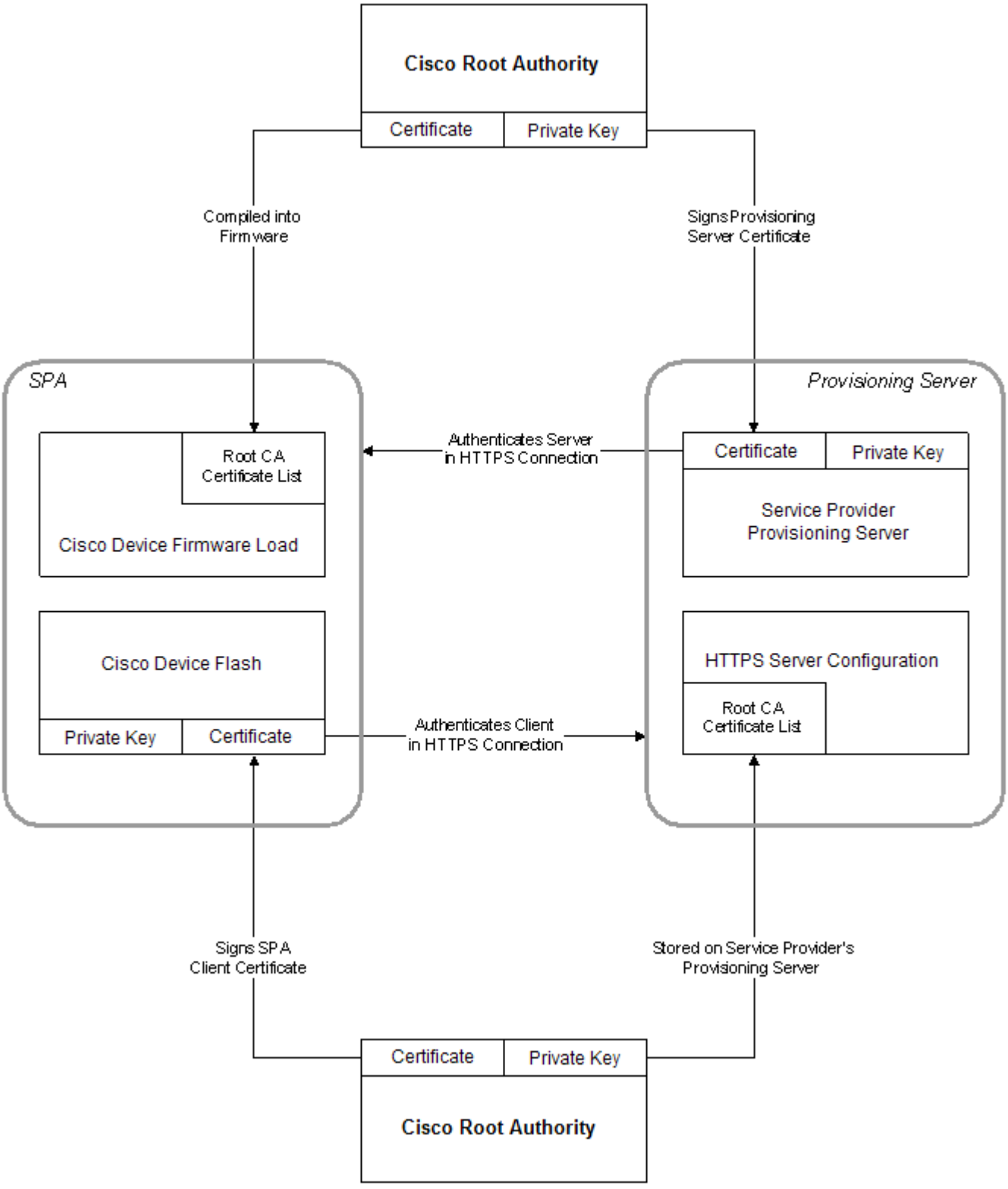
In addition to a direct attack on an IP Telephony Device, an attacker might attempt to contact a provisioning server by using a standard web browser or other HTTPS client, to obtain the configuration profile from the provisioning server. To prevent this kind of attack, each IP Telephony Device also carries a unique client certificate, also signed by Cisco, including identifying information about each individual endpoint. A certificate authority root certificate capable of authenticating the device client certificate is given to each service provider. This authentication path allows the provisioning server to reject unauthorized requests for configuration profiles.

Certificate Structure

The combination of a server certificate and a client certificate ensures secure communication between a remote IP Telephony Device and its provisioning server. The **"Certificate Authority Flow"** figure illustrates the relationship and placement of certificates, public/private key pairs, and signing root authorities, among the Cisco client, the provisioning server, and the certification authority.

The upper half of the diagram shows the Provisioning Server Root Authority, which is used to sign the individual provisioning server certificate. The corresponding root certificate is compiled into the firmware, allowing the IP Telephony Device to authenticate authorized provisioning servers.

Certificate Authority Flow



As indicated in the lower half of the diagram, a Cisco Small Business Client Certificate Root Authority signs each unique certificate. The corresponding root certificate is made available to service providers for client authentication purposes.

Provisioning Setup

This section describes setup requirements for provisioning an IP Telephony Device and includes the following topics:

Software Tools

The following software tools are useful for provisioning Cisco Small Business IP Telephony Devices :

- Open source gzip compression utility, used when generating configuration profiles
- Open source OpenSSL software package: for profile encryption and HTTPS operations
- Scripting language with CGI scripting support, such as the open source Perl language tools: to test dynamic generation of profiles and one-step remote provisioning using HTTPS
- Ethernet packet analyzer (such as the freely downloadable Ethereal/ Wireshark): to verify secure exchanges between provisioning servers and Cisco Small Business voice devices
- The ssldump utility: for monitoring HTTPS transactions

Server Configuration

Provisioning requires the availability of servers, which for testing purposes can be installed and run on a local PC:

- TFTP (UDP port 69)
- HTTP (TCP port 80)
- HTTPS (TCP port 443)
- Syslog (UDP port 514)

To troubleshoot server configuration, it is helpful to install a separate client for each type of server on a different host.

TFTP

TFTP is convenient for managing small deployments of IP Telephony Devices within an office LAN environment. It is also useful for in-house preprovisioning of IP Telephony Devices in preparation for remote deployment. However, once deployed remotely, HTTP offers greater provisioning reliability, given NAT and router protection mechanisms.

The IP Telephony Device is able to obtain a TFTP server IP address directly from the DHCP server through DHCP option 66. If this is done, a Profile_Rule need be configured only with the profile filepath on that TFTP server. The Profile_Rule provided with the factory default configuration is as follows: */device*.cfg

For example, on a SPA962, the filename is spa962.cfg. The device resyncs to this file on the local TFTP server, if that is specified via DHCP option 66. Note that the specified filepath is relative to the TFTP server virtual root directory.

HTTP

The IP Telephony Device behaves like a browser requesting web pages from any remote Internet site. This provides a reliable means of reaching the provisioning server, even when a customer router implements symmetric NAT or other protection mechanisms. HTTP and HTTPS works more reliably than TFTP in remote deployments, especially when the deployed units are connected behind residential firewalls or NAT-enabled routers.

As an alternative to HTTPS, the IP Telephony Device can resync to a configuration profile using HTTP. In this case, a separate explicit profile encryption can be used to protect confidential information. The IP Telephony Device supports 256-bit AES in CBC mode to pre-encrypt individual profiles. These encrypted profiles can be downloaded by the IP Telephony Device using HTTP without danger of unauthorized use of confidential information in the configuration profile. This resync mode may be useful to reduce the computational load on the provisioning server required when using HTTPS for every resync request.

In a small deployment within a single LAN environment, it is common to rely on a simple TFTP server for provisioning of network devices. Cisco Small Business voice devices support TFTP for both provisioning resync and firmware upgrade operations. TFTP is especially useful for the in-house preprovisioning of a large number of un-provisioned devices.

Basic HTTP-based provisioning relies on the HTTP GET method for retrieving configuration profiles. Typically, this means that a configuration file is pre-generated for each deployed IP Telephony Device, and these files are stored within an HTTP server directory. When the server receives the GET request, it simply returns the file specified in the GET request header.

Alternatively, the requested URL can invoke a CGI script (still using the GET method). In this case, the configuration profile might be generated dynamically, perhaps by querying a customer database and producing the profile on-the-fly.

In the case of CGI handling resync requests, the IP Telephony Device also supports the HTTP POST method as a mechanism to request the resync configuration data. The device can be configured to convey certain status and identification information to the server within the body of the HTTP POST request. The server can use this information to help generate a desired response configuration file, or store the status information for later analysis and tracking.

As part of both GET and POST requests, the IP Telephony Device automatically includes basic identifying information in the request header, in the User-Agent field. The supplied information conveys manufacturer, product name, current firmware version, and product serial number.

For example, the following is the User-Agent request field from a SPA962:

```
User-Agent: cisco/SPA-962-2.0.5 (88012BA01234)
```

Enabling HTTPS

For increased security managing remotely deployed units, the IP Telephony Device supports HTTPS for provisioning. To this end, each newly manufactured IP Telephony Device carries a unique SLL Client Certificate (and associated private key), in addition to a Sipura CA server root certificate. The latter allows the IP Telephony Device to recognize authorized provisioning servers, and reject non-authorized servers. On the other hand, the client certificate allows the provisioning server to identify the individual device that issues the request.

In order for a service provider to manage deployment by using HTTPS, a server certificate needs to be generated for each provisioning server to which the IP Telephony Device resyncs using HTTPS. The server certificate must be signed by the Cisco Server CA Root Key, whose certificate is carried by all deployed units. To obtain a signed server certificate, the service provider must forward a certificate signing request to Cisco, which signs and returns the server certificate for installation on the provisioning server.

The provisioning server certificate must contain in the subject Common Name (CN field) the FQDN of the host running the server. It may optionally contain additional information following the host FQDN, separated by a / character. The following are examples of CN entries that would be accepted as valid by the IP Telephony Device:

```
CN=sprov.callme.com
CN=pv.telco.net/mailto:admin@telco.net
CN=prof.voice.com/info@voice.com
```

In addition to verifying the server certificate, the IP Telephony Device tests the server IP address against a DNS lookup of the server name specified in the server certificate.

A certificate signing request can be generated using the OpenSSL utility. The following shows an example of the **openssl** command that produces a 1024-bit RSA public/private key pair and a certificate signing request:

```
openssl req –new –out provserver.csr
```

This command generates the server private key in privkey.pem and a corresponding certificate signing request in provserver.csr. In this example, the service provider keeps privkey.pem secret and submits provserver.csr to Cisco for signing. Upon receiving the provserver.csr file, Cisco generates provserver.crt, the signed server certificate.

In addition, Cisco also provides a Sipura CA Client Root Certificate to the service provider. This root certificate certifies the authenticity of the client certificate carried by each IP Telephony Device.

The unique client certificate offered by each device during an HTTPS session carries identifying information embedded in its subject field. This information can be made available by the HTTPS server to a CGI script invoked to handle secure requests. In particular, the certificate subject indicates the unit product name (OU element), MAC address (S element), and serial number (L element). The following is an example of these elements from a SPA962 client certificate subject field:

```
OU=SPA-962, L=88012BA01234, S=000e08abcdef
```

Early units, manufactured before firmware 2.0.x, do not contain individual SSL client certificates. When these units are upgraded to a firmware release in the 2.0.x tree, they become capable of connecting to a secure server using HTTPS, but are only able to supply a generic client certificate if requested to do so by the server. This generic certificate contains the following information in the identifying fields:

```
OU=cisco.com, L=ciscogeneric, S=ciscogeneric
```

To determine if an IP Telephony Device carries an individualized certificate use the $CCERT provisioning macro variable, whose value expands to either Installed or Not Installed, according to the presence or absence of a unique client certificate. In the case of a generic certificate, it is possible to obtain the serial number of the unit from the HTTP request header, in the User-Agent field.

HTTPS servers can be configured to request SSL certificates from connecting clients. If enabled, the server can verify the client certificate by using the Sipura CA Client Root Certificate supplied by Cisco. It can then provide the certificate information to a CGI for further processing.

The location for storing certificates may vary. For example, on a Apache installation, the file paths for storing the provisioning server signed certificate, its associated private key, and the Sipura CA client root certificate are likely to be as follows:

```
# Server Certificate:
SSLCertificateFile /etc/httpd/conf/provserver.crt

# Server Private Key:
SSLCertificateKeyFile /etc/httpd/conf/provserver.key

# Certificate Authority (CA):
SSLCACertificateFile /etc/httpd/conf/spacroot.crt
```

Refer to the documentation provided for an HTTPS server for specific information.

Firmware release 2.0.6 supports the following cipher suites for SSL connection to a server using HTTPS. Future release updates may implement additional cipher suites.

Table 1 Cipher Suites Supported for Connecting to an HTTPS Server

Numeric Code	Cipher Suite
0x0039	TLS_DHE_RSA_WITH_AES_256_CBC_SHA
0x0035	TLS_RSA_WITH_AES_256_CBC_SHA
0x0033	TLS_DHE_RSA_WITH_AES_128_CBC_SHA
0x002f	TLS_RSA_WITH_AES_128_CBC_SHA
0x0005	TLS_RSA_WITH_RC4_128_SHA
0x0004	TLS_RSA_WITH_RC4_128_MD5
0x0062	TLS_RSA_EXPORT1024_WITH_RC4_56_SHA
0x0060	TLS_RSA_EXPORT1024_WITH_RC4_56_MD5
0x0003	TLS_RSA_EXPORT_WITH_RC4_40_MD5

Syslog Server

If a syslog server is configured on the IP Telephony Device (using the <Syslog_Server> or <Debug_Server> parameters), the resync and upgrade operations log messages to the syslog server. A message can be generated at the start of a remote file request (configuration profile or firmware load), and at the conclusion of the operation (with either success or failure).

The logged messages themselves are configured in the following parameters:

- For profile resync:
  - Log_Resync_Request_Msg
  - Log_Resync_Success_Msg
  - Log_Resync_Failure_Msg

- For firmware upgrades:
  - Log_Upgrade_Request_Msg
  - Log_Upgrade_Success_Msg
  - Log_Upgrade_Failure_Msg

These parameters are macro expanded into the actual syslog messages.

Where to Go From Here

The following table summarizes the location of specific information in this document for completing different provisioning tasks.

To do this...	Refer to...
Learn to work with Cisco provisioning scripts and configuration profiles.	**Chapter 2, "Creating Provisioning Scripts"**
Review step-by-step procedures for using the scripting language to create a configuration profile.	**Chapter 3, "Provisioning Tutorial"**
Refer to the function and usage of each parameter on the Provisioning tab of the administration web server.	**Chapter 4, "Provisioning Field Reference"**
View a sample profile.	**Appendix A, "Example Configuration Profile"**
Look up the expansion for an acronyms use in this document.	**Appendix B, "Acronyms"**

2

Creating Provisioning Scripts

This chapter describes the provisioning script and includes the following sections:

NOTE For detailed information about your IP Telephony Devices, refer to the *SPA9000 Administration Guide*, the *ATA Administration Guide*, the *WRP400 User Guide*, or the *SPA and Wireless IP Phone Administration Guide*. Each guide describes the parameters that can be configured through the administration web server, which likewise are used in configuration profiles.

Configuration Profile and the SIP Profile Compiler

The configuration profile defines the parameter values for the IP Telephony Device. You can use the profile to set the value for each parameter that is used by the IP Telephony Device. You also can specify the user access to each parameter: hidden, read-only, or read-write. Any parameters that are not specified by a profile are left at the factory default values.

The IP Telephony Device accepts a configuration profile in two formats:

- Open Format (XML-style). See **Open Format Configuration File, page 31**.

- Proprietary Plain-Text Format. See **Proprietary Plain-Text Configuration File, page 40**.

  You will use the SIP Profile Compiler to create your configuration profile. You can use SPC to to generate an XML file that contains all the existing settings for a particular device, and then you can edit the file as needed to input the new settings. Alternatively, if you are using the proprietary plain-text format, you can use the SPC to compile your text file as a binary object.

  To download the SPC tool, see see **Downloading the SIP Profile Compiler (SPC) Tool, page 17**. To run the SPC tool, open a command prompt, and then run the executable file.

Open Format Configuration File

The XML-style format lets you use standard tools to create the configuraqtion file. A configuration file in open, XML-style format can be sent from the provisioning server to the IP Telephony Device during a resync operation without compiling the file as a binary object.

To protect confidential information contained in the configuration profile, this file is generally delivered from the provisioning server to the IP Telephony Device over a secure channel, provided by HTTPS. A complete example XML profile can be generated by using the SIP Profile Compiler tool.

The IP Telephony Device can accept configuration formats that are generated by standard tools. This feature eases development of back-end provisioning server software to generate configuration profiles from existing databases.

This open format consists of a text file with XML-like syntax. Optionally, the file can be compressed by using the gzip deflate algorithm (RFC1951). In addition, the file can be encrypted by using 256-bit AES symmetric key encryption.

The XML profile syntax consists of an XML-style hierarchy of elements, with element attributes and values. Refer to the following example and the notes below it.

Example: Basic XML Profile Format

```
<flat-profile>
<Resync_On_Reset> Yes
  </Resync_On_Reset>
<Resync_Periodic> 7200
  </Resync_Periodic>
<Profile_Rule>
```

```
    tftp://prov.telco.com:6900/cisco/config/spa962.cfg
  </Profile_Rule>
</flat-profile>
```

A file can include element tags, attributes, and formatting features.

Element tags:

- The IP Telephony Device recognizes elements with proper parameter names, when encapsulated in the special <flat-profile> element.
- The <flat-profile> element can be encapsulated within other arbitrary elements.
- Enclose element names in angle brackets.
- The element names derive from the field names in the administration web pages for the device, with the following modifications:
  - Element names may not include spaces or special characters. To derive the element name from the field name, subsitute an underscore for spaces and the following special characters: `[ ] ( ) /`

    For example, the Resync On Reset field is represented by the following element: `<Resync_On_Reset>`
  - Each element name must be unique. In the administration web pages, the same fields may appear on multiple web pages, such as the Line, User, and Extension pages. Append [*n*] to the element name to indicate the number that is shown in the page tab.

    For example, the Dial Plan for Line 1 is represented by the following element: `<Dial_Plan[1]>`
- Make sure that each opening element tag is properly matched by a corresponding closing element tag. Refer to the following example:

```
<flat-profile>
<Resync_On_Reset> Yes
    </Resync_On_Reset>
<Resync_Periodic> 7200
    </Resync_Periodic>
<Profile_Rule>tftp://prov.telco.com: 6900/cisco/config/
spa962.cfg
    </Profile_Rule>
```

```
</flat-profile>
```

- Element tags are case sensitive.
- Empty element tags are allowed.
- Unrecognized element names are ignored.
- You can use an empty element tag to prevent the overwriting of any user-supplied values during a resync operation. In the following example, the user speed dial settings are unchanged.

```
<Speed_Dial_2_2_  ua="rw"/>
  <Speed_Dial_3_2_  ua="rw"/>
  <Speed_Dial_4_2_  ua="rw"/>
  <Speed_Dial_5_2_  ua="rw"/>
  <Speed_Dial_6_2_  ua="rw"/>
  <Speed_Dial_7_2_  ua="rw"/>
  <Speed_Dial_8_2_  ua="rw"/>
  <Speed_Dial_9_2_  ua="rw"/>
</flat-profile>
```

- You can enter an empty value to set the corresponding parameter to an empty string. Enter an opening and closing element without any value between them. In the following example, the GPP_A parameter is set to an empty string.

```
<flat-profile>
<GPP_A>
  </GPP_A>
</flat-profile>
```

Attributes:

- Element attributes are allowed. Their value must be enclosed by double quotes. All such attributes are ignored by the IP Telephony Device, except for the user-access attribute: ua.

- The user-access attribute defines access to the administration web server for a specific parameter by the User account. Access by the Admin account is unaffected by this attribute.
- The ua attribute, if present, must have one of the following values:
  - na—no access
  - ro—read-only
  - rw—read/write

  The ua attribute is illustrated by the following example:

```
<flat-profile>
  <SIP_TOS_DiffServ_Value_1_   ua="na"/>
  <Dial_Plan_1_   ua="ro"/>
  <Dial_Plan_2_   ua="rw"/>
</flat-profile>
```

- If the user-access attribute (ua) is not specified in an element tag, the factory default user access is applied for the corresponding parameter.

Parameters:

- Any parameters that are not specified by a profile are left unchanged in the IP Telephony Device.
- Unrecognized parameters are ignored.
- The IP Telephony Device recognizes arbitrary, configurable aliases for a limited number of parameter names.
- If the XML file contains multiple occurrences of the same parameter tag, the last such occurrence overrides any earlier ones. To avoid inadvertently overriding configuration values for a parameter, it is recommended that at most one instance of a parameter be specified in any one profile.

Formatting:

- Comments are allowed, using standard XML syntax.

```
<!-- My comment is typed here -->
```

- Leading and trailing white space is allowed for readability and will be removed from the parameter value.
- New lines within a value are converted to spaces.

- An XML header of the form <? . . . ?> is allowed, but is ignored by the IP Telephony Device.
- To enter special characters, use basic XML character escapes, as shown in the following table.

Special Character	XML Escape Sequence
& (ampersand)	&amp;
< (less than)	&lt;
> (greater than)	&gt;
’ (apostrophe)	&apos;
” (double quote)	&quot;

In the following example, character escapes are entered to represent the greater than and less than symbols that are required in a dial plan rule. This example defines an information hotline dial plan, which sets the Dial_Plan[1] parameter equal to (S0 <:18005551212>).

```
<flat-profile>
 <Dial_Plan_1_>
    ( S0 &lt;:18005551212&gt; )
  </Dial_Plan_1_>
</flat-profile>
```

NOTE

- Numeric character escapes, using decimal and hexadecimal values (s.a. `&#40;` and `&#x2e;`), are also translated.
- The firmware does not support the full Unicode character set, but only the ASCII subset.

Configuration File Compression

Optionally, the XML configuration profile can be compressed to reduce the network load on the provisioning server. The supported compression method is the gzip deflate algorithm (RFC1951). The gzip utility and a compression library that implements the same algorithm (zlib) are readily available from Internet sites.

To identify when compression is applied, the IP Telephony Device expects the compressed file to contain a gzip compatible header, as generated by invoking the gzip utility on the original XML file.

For example, if profile.xml is a valid profile, the file profile.xml.gz is also accepted. This example can be generated with either of the following commands:

```
# first invocation, replaces original file with compressed file:

gzip profile.xml

# second invocation, leaves original file in place, produces new compressed
file:

cat profile.xml | gzip > profile.xml.gz
```

The IP Telephony Device inspects the downloaded file header to determine the format of the file. The choice of file name is not significant and any convention that is convenient for the service provider can be used.

File Encryption

An XML configuration profile can be encrypted using symmetric key encryption, whether or not it is already compressed. The supported encryption algorithm is the American Encryption Standard (AES), using 256-bit keys, applied in cipher block chaining mode.

NOTE Compression must precede encryption for the IP Telephony Device to recognize a compressed and encrypted XML profile. First generate the XML, then compress with gzip, and finally encrypt.

The OpenSSL encryption tool, available for download from various Internet sites, can be used to perform the encryption. Note that support for 256-bit AES encryption may require recompilation of the tool (so as to enable the AES code). The firmware has been tested against version openssl-0.9.7c.

If the file is encrypted, the profile expects the file to have the same format as generated by the following command:

```
# example encryption key = SecretPhrase1234

openssl enc –e –aes-256-cbc –k SecretPhrase1234 –in profile.xml –out
profile.cfg

# analogous invocation for a compressed xml file

openssl enc –e –aes-256-cbc –k SecretPhrase1234 –in profile.xml.gz –out
profile.cfg
```

A lower case –k precedes the secret key, which can be any plain text phrase and is used to generate a random 64-bit salt. Then, in combination with the secret specified with the –k argument, it derives a random 128-bit initial vector, and the actual 256-bit encryption key.

When this form of encryption is used to encrypt a configuration profile, the IP Telephony Device needs to be informed of the secret key value to decrypt the file. This value is specified as a qualifier in the pertinent profile URL. The syntax is as follows, using an explicit URL:

```
[--key “SecretPhrase1234”] http://prov.telco.com/path/profile.cfg
```

This value is programmed using one of the Profile_Rule parameters. The key must be preprovisioned into the unit at an earlier time. This bootstrap of the secret key can be accomplished securely by using HTTPS.

Preencrypting configuration profiles offline with symmetric key encryption allows the use of HTTP for resyncing profiles. The provisioning server uses HTTPS to handle initial provisioning of IP Telephony Devices after deployment. This feature reduces the load on the HTTPS server in large scale deployments.

The final file name does not need to follow a specific format, but it is conventional to end the name with the .cfg extension to indicate that it is a configuration profile.

Encrypting a File with the SPC

The SPC can generate different types of configuration files, using different types of encryption.

- Generic, non-targeted CFG file, without an explicit key
- Targeted (**--target** option), also encrypts the CFG file without an explicit key, but uses the MAC address of the target IP Telephony Device, and only that device can decode it
- Explicit key-based encryption of the CFG file.

A generic, non-targeted CFG file is accepted as valid by any IP Telephony Device that resyncs to it. The following command generates a basic CFG file:

```
spc spa962.txt spa962.cfg
```

This example compiles the plain-text spa962.txt file into the binary spa962.cfg file understood by the SPA962. The **--scramble** option performs encryption that does not require the explicit transmission of a key to the target device. It requires one randomizing argument. For example,

```
spc --scramble SomeSecretPhrase spa962.txt spa962.cfg
```

The resulting encrypted spa962.cfg is accepted as valid by any IP Telephony Device that resyncs to it.

The **--target** option also encrypts the CFG file without the need to explicitly transmit a key, but does so in such a way that only the target IP Telephony Device can decode it. Targeted CFG files provide a basic level of security. This command uses the MAC address of the target device as an argument. Refer to the following example:

```
spc --target 000e08aabbcc spa962.txt spa962.cfg
```

This command uses the MAC address 000e08aabbcc, and only the IP Telephony Device with that MAC address is able to decrypt and process the generated spa962.cfg profile. If any other IP Telephony Device attempts to resync to this file, the device will reject the file as unreadable.

The third option performs an explicit key-based encryption of the CFG file. This option requires that the key used to encrypt the file be preprovisioned in the target device, so that it can be decoded.

Two algorithms are available for this type of encryption:

- RC4 (**--rc4**)
- AES (**--aes**)

In addition, the key can be specified either explicitly as a hexadecimal digit sequence (**--hex-key**) or by hashing a secret phrase (**--ascii-key**). With the **--hex-key** option, the key can be up to 256 bits in length. With the **--ascii-key** option the generated key is 128 bits.

The following examples illustrate explicit key-based encryption.

```
spc –-rc4 –-ascii-key apple4sale spa962.txt spa962.cfg
spc –-aes –-ascii-key lucky777 spa962.txt spa962.cfg
spc –-aes –-ascii-key “my secret phrase” spa962.txt spa962.cfg
spc –-aes –-hex-key 8d23fe7...a5c29 spa962.txt spa962.cfg
```

Any combination of scrambling, targeting, and explicit-key encrypting can be applied to a CFG file, as shown by the following example:

```
spc –-target 000e08aaa010 –-aes –-ascii-key VerySecret a.txt a.cfg
```

After each compilation, SPC prints a final status message. Syntax error messages are also printed if a compilation is not successful.

The status and error messages printed by SPC are suppressed with the **--quiet** command line option. Messages can be redirected to a file with the **--log** *file_name* option. In the latter case, the SPC command itself is also printed in the log file, preceded by a timestamp.

```
spc –-quiet . . .
spc –-log prov.log . . .
```

SPC can also be used to generate sample configuration source files (for both plain text and XML formats), corresponding to the accompanying firmware release. The commands for producing sample files are as follows:

```
# sample plain.txt to be used as source file for eventual spc compilation:

spc –-sample-profile plain.txt

# sample config.xml to be fed directly to a device running 2.0.6 or above:

spc --sample-xml config.xml
```

Proprietary Plain-Text Configuration File

The plain-text configuration file uses a proprietary format, which can be encrypted to prevent unauthorized use of confidential information. By convention, the profile is named with the extension .cfg (for example, spa962.cfg). The SPC tool is used to compile the plain-text file into an encrypted CFG file.

The plain-text format is an alternative to the open format and is the only format recognized by firmware releases prior to 2.0.6.

Source Text Syntax

The syntax of the plain-text file accepted by SPC is a series of parameter-value pairs, with the value enclosed in double quotes. Each parameter-value pair is followed by a semicolon (for example, parameter_name “parameter_value”;). If no quoted value is specified for a parameter (or if a parameter specification is missing entirely from the plain-text file) the value of the parameter remains unchanged in the IP Telephony Device.

The syntax also controls the User account access to the parameter in the administration web server. An optional exclamation point or question mark, immediately following the parameter name, indicates the parameter should be read-write or user read-only for the User account.

If neither mark is present, the parameter is made inaccessible to the user from the web server pages. Note that this syntax has no effect on the Admin account access to the parameter. If the parameter specification is missing entirely from the plain-text file, the User account access to the parameter remains unchanged in the IP Telephony Device.

If the plain-text file contains multiple occurrences of the same parameter-value specification, the last occurrence overrides any earlier ones. To avoid accidentally overwriting configuration values, it is recommended that a profile includes no more than one specification for each parameter.

The element names derive from the field names in the administration web pages for the device, with the following modifications:

- Element names may not include spaces or special characters. To derive the element name from the field name, subsitute an underscore for spaces and the following special characters: `[ ] ( ) /`

  For example, the Resync On Reset field is represented by the following element: `<Resync_On_Reset>`

- Each element name must be unique. For fields that are duplicated on multiple Line, User, or Extension pages, you must append [*n*] to indicate the line, user, or extension number.

  For example, the Dial Plan for Line 1 is represented by the following element: `<Dial_Plan[1]>`

The following additional features can be used:

- Comments are delimited by a # character up to the end-of-line.
- Blank lines can be used for readability.

The following illustrates the format for each parameter-value pair:

```
Parameter_name [ ‘?’ | ‘!’ ] [“quoted_parameter_value_string”] ‘;’
```

Boolean parameter values are asserted by any one of the values {Yes | yes | Enable | enable | 1}. They are deasserted by any one of the values {No | no | Disable | disable | 0}.

The following are examples of plain-text file entries:

```
# These parameter names are for illustration only

Feature_Enable      ! “Enable” ;   # user read-write, but force the value to
Enable
Another_Parameter   ? “3600”   ;   # user read-only
Hidden_Parameter      “abc123” ;   # user not-accessible

Some_Entry          !          ;   # user read-write, leaves value unchanged
```

Multiple plain text files can be spliced together to generate the source for the final binary CFG file. This is accomplished using the **import** directive at the start of a new line followed by one or more spaces and the file name to splice into the stream of parameter-value pairs. File splicing can be nested several files deep.

For example, the file base.txt contains the following:

```
Param1 “base value 1” ;
Param2 “base value 2” ;
```

The file spa1234.txt contains the following lines:

```
import base.txt
Param1 “new value overrides base” ;
Param7 “particular value 7” ;
```

When compiled, spa1234.txt becomes:

```
Param1 “base value 1” ;
Param2 “base value 2” ;
Param1 “new value overrides base” ;
Param7 “particular value 7” ;
```

Comments

During development and scripting, it is often convenient to temporarily disable a provisioning parameter by entering a # character at the start of the parameter value. This effectively comments-out the remaining text in that parameter.

For example, a Profile_Rule with the value `“# http://192.168.1.200/sample.cfg”` is equivalent to an empty Profile_Rule. The # character comment-mechanism applies to the Profile_Rule*, Upgrade_Rule, and Resync_Trigger_* parameters.

Macro Expansion

Several provisioning parameters undergo macro expansion internally prior to being evaluated. This preevaluation step provides greater flexibility controlling the resync and upgrade activities of the IP Telephony Device.

The parameter groups which undergo macro expansion before evaluation are as follows:

- Resync_Trigger_*
- Profile_Rule*
- Log_Resync_*
- Upgrade_Rule
- Log_Upgrade_*

Under certain conditions, some general purpose parameters (GPP_*) also undergo macro expansion, as explicitly indicated in the Optional Resync Arguments section.

During macro expansion, expressions of the form $NAME and $(NAME) are replaced by the contents of the named variables. These variables include general purpose parameters, several product identifiers, certain event timers, and provisioning state values. For a complete list, see the **"Macro Expansion Variables" section on page 91**.

In the following example, the expression $(MAU) is used to insert the MAC address 000E08012345.

> The administrator enters: `spa$(MAU)config.cfg`
> The resulting macro expansion for a device with MAC address 000E08012345 is: `spa000E08012345config.cfg`

If a macro name is not recognized, it remains unexpanded. For example, the name STRANGE is not recognized as a valid macro name, while MAU is recognized as a valid macro name.

> The administrator enters: `spa$STRANGE$MAU.cfg`
> The resulting macro expansion for a device with MAC address 000E08012345 is: `spa$STRANGE000E08012345.cfg`

Macro expansion is not applied recursively. For example, $$MAU" expands into $MAU" (the $$ is expanded), and does not result in the MAC addres.

The special purpose parameters (GPP_SA through GPP_SD), whose contents are mapped to the macro expressions $SA through $SD, are only macro expanded as the argument of the **--key** option in a resync URL.

Also, the macro expression can qualify the expansion so that only a substring of the macro variable is used instead of its full value, such as a portion of the MAC address.

The syntax for substring macro expansion is $(NAME:p) and $(NAME:p:q), where p and q are non-negative integers. The resulting expansion results in the macro variable substring starting at character offset p, and of length q (or till end-of-string if q is not specified). Refer to the following examples.

> The administrator enters: `$(MAU:4)`
> The resulting macro expansion for a device with MAC address 000E08012345 is: `08012345`

> The administrator enters: `$(MAU:8:2)`
> The resulting macro expansion for a device with MAC address 000E08012345 is: `23`

Conditional Expressions

Conditional expressions can trigger resync events and select from alternative URLs for resync and upgrade operations.

Conditional expressions consist of a list of comparisons, separated by the **and** operator. All comparisons must be satisfied for the condition to be true.

Each comparison can relate one of three types of literals:

- Integer values
- Software or hardware version numbers
- Doubled-quoted strings

Note that version numbers take the form of three non-negative integers separated by periods (major, minor, and build numbers), plus an optional alphanumeric string in parentheses. No intervening spaces are allowed.

The following are examples of valid version numbers:

```
1.0.31(b)
1.0.33
2.0.3(G)
2.0.3(0412s)
2.0.6
```

Quoted strings can be compared for equality or inequality. Integers and version numbers can also be compared arithmetically. The comparison operators can be expressed as symbols or as acronyms, as indicated in the table below. Acronyms are particularly convenient when expressing the condition in an XML-style profile.

Operator	Alternate Syntax	Description	Applicable to Integer and Version Operands	Applicable to Quoted String Operands
=	eq	equal to	Yes	Yes
!=	ne	not equal to	Yes	Yes
<	lt	less than	Yes	No
<=	le	less than or equal to	Yes	No

Operator	Alternate Syntax	Description	Applicable to Integer and Version Operands	Applicable to Quoted String Operands
>	gt	greater than	Yes	No
>=	ge	greater than or equal to	Yes	No

For legacy support to firmware versions prior to 2.0.6, the not-equal-to operator can also be expressed as a single ! character (in place of the two-character != string).

Conditional expressions typically involve macro-expanded variables. For example:

```
$REGTMR1 gt 300 and $PRVTMR gt 1200 and “$EXTIP” ne “”

$SWVER ge 2.0.6 and “$CCERT” eq “Installed”
```

It is important to enclose macro variables in double quotes where a string literal is expected. Do not do so where a number or version number is expected.

For legacy support of firmware versions prior to 2.0.6, a relational expression with no left-hand-side operand assumes $SWVER as the implicit left-hand-side. For example, ! 1.0.33 is equivalent to: $SWVER != 1.0.33.

When used in the context of the Profile_Rule* and Upgrade_Rule parameters, conditional expressions must be enclosed within the syntax “(expr)?” as in the following upgrade rule example:

```
( $SWVER ne 2.0.6 )? http://ps.tell.com/sw/spa021024.bin
```

On the other hand, the syntax above using parentheses should not be used when configuring the Resync_Trigger_* parameters.

Assignment Expressions

Arbitrary parameters can be pre-assigned values within the context of Profile_Rule* and Upgrade_Rule parameter. This causes the assignment to be performed before the profile if retrieved.

The syntax for performing these assignments is a list of individual parameter assignments, enclosed within parentheses (assignments)!, with each assignment taking the form:

ParameterXMLName = “Value” ;

Note that the recognized parameter names correspond to the names as for XML-based profiles.

Any parameter can be assigned a new value in this way, and macro-expansion applies. For example, the following is a valid assignment expression:

```
( User_ID_1_ = “uid$B” ; GPP_C = “” ; GPP_D = “$MA” ; )!
```

For conciseness, the general purpose parameters GPP_A through GPP_P can also be referred to by the single lowercase letters a through p. The example above is equivalent to the following:

```
( User_ID_1_ = “uid$B” ; c = “” ; d = “$MA” ; )!
```

White space can optionally be used for readability.

URL Syntax

Standard URL syntax is used to specify how to retrieve configuration files and firmware loads in Profile_Rule* and Upgrade_Rule parameters, respectively. The syntax is as follows:

```
[ scheme:// ] [ server [:port]] filepath
```

Where scheme is one of the following values:

- tftp
- http
- https

If scheme is omitted, tftp is assumed. The server can be a DNS-recognized host name or a numeric IP address. The port is the destination UDP or TCP port number. The filepath must begin with the root directory (/). In other words, it must be an absolute path.

If server is missing, then the tftp server specified through DHCP (option 66) is used instead.

If port is missing, then the standard port for the specified scheme is used instead (tftp uses UDP port 69, http uses TCP port 80, https uses TCP port 443). A filepath must be present. It need not necessarily refer to a static file, but can indicate dynamic content obtained through CGI.

Macro expansion applies within URLs. The following are examples of valid URLs:

```
/$MA.cfg
/cisco/spa021025.bin
192.168.1.130/profiles/init.cfg
tftp://prov.call.com/cpe/cisco$MA.cfg
http://neptune.speak.net:8080/prov/$D/$E.cfg
https://secure.me.com/profile?Linksys
```

Optional Resync Arguments

The URLs entered in Profile_Rule* parameters may be preceded by optional arguments, collectively enclosed by square brackets. The recognized options are key, post, and alias.

key

The **key** option is used to specify an encryption key. It is required to decrypt profiles which have been encrypted with an explicit key. The key itself is specified as a (possibly quoted) string following the term **--key**.

Some usage examples:

```
[--key VerySecretValue]
[--key “my secret phrase”]
[--key a37d2fb9055c1d04883a0745eb0917a4]
```

The bracketed optional arguments are macro expanded. In particular, note that the special purpose parameters GPP_SA through GPP_SD are only macro expanded into their macro variables $SA through $SD when used as arguments of the key option, as in the following examples:

```
[--key $SC]
[--key “$SD”]
```

In the case of XML-style profiles, the argument to **--key** must be the same as the argument to the **-k** option given to **openssl**.

In the case of SPC compiled profiles, the argument to **--key** must be the same as the argument to either the **--ascii-key** or the **--hex-key** options, as given to SPC.

post

The **post** option provides an alternative access method for the http and https schemes. If left unspecified, the IP Telephony Device performs an HTTP GET operation, when contacting the provisioning server. If specified, on the other hand, the device performs an HTTP POST operation.

The body of the POST is generated from the contents of one of the general purpose parameters, GPP_A through GPP_P, with macro expansion applied. The GPP_* parameter to use is indicated by a single lowercase letter (a through p) given as argument to the term **--post**.

Using POST provides a convenient alternative to the GET method when arbitrary state or identifying information needs to be supplied from the IP Telephony Device to the server, as part of periodic resyncs.

For example, GPP_F could contain the following POST body template:

```
Product = “$PN”; MAC_Addr = “$MA”; Ser_Num = “$SN”; SW_Ver = “$SWVER”;
```

Then, a URL option such as the following would use the POST method to convey the information to the server in the body of the profile request message (shown here with an accompanying URL):

```
[--post f ] http://ps.one.com/cpe/resyncs?
```

alias

The **alias** option provides a flexible means of recognizing alternative parameter names in XML-based configuration profiles. This is useful in cases where part of the configuration profile is obtained from a customer database form that uses different terminology than expected by the IP Telephony Device.

For example, a customer XML profile specifies the SIP registration parameters: name, number, auth-secret, enclosed in an XML element hierarchy as follows:

```
<CPE>
  <SIP-Credentials>
    <name>J. Smith</name>
    <number>14085551234</number>
    <auth-secret>732091751563sfd</auth-secret>
```

```
  <SIP-Credentials>
</CPE>
```

To map these three parameters directly to the Display_Name_1_, User_ID_1_, and Password_1_ parameters (Line 1), enter this mapping in a general purpose parameter (for example, GPP_M):

```
/CPE/SIP-Credentials/name = /flat-profile/Display_Name_1_ ;
/CPE/SIP-Credentials/number = /flat-profile/User_ID_1_ ;
/CPE/SIP-Credentials/auth-secret = /flat-profile/Password_1_ ;
```

Then, request the customer credentials profile with the following URL option (showing an example URL for completeness):

```
[--alias m ] http://acct.voipservice.net/credentials/spa$MA.xml
```

Upon receiving the profile, the IP Telephony Device would apply the indicated translations, assigning J. Smith to Display_Name_1_, 14085551234 to User_ID_1_, and 732091751563sfd to Password_1_.

The **alias** option matches only the left-hand-side of an alias as much as specified by the configured alias map. The element itself can be nested further. In the example above, GPP_M could have contained the following instead:

```
/SIP-Credentials/name = /flat-profile/Display_Name_1_ ;
/SIP-Credentials/number = /flat-profile/User_ID_1_ ;
/auth-secret = /flat-profile/Password_1_ ;
```

In general, it is best to specify enough enclosing elements to ensure an unambiguous translation.

The **alias** option is designed to recognize a limited number of critical parameters. Up to 30 parameters can be remapped this way.

Combining Options

Multiple URL options can be combined, by enclosing them within the same set of square brackets. The following are examples of valid URL option strings:

```
[--post j --alias k]
[--key “SymmetricSecret” --alias a]
[--key “$SB” --post g]
[--alias a --key abracadabra321 --post c]
```

Using Provisioning Parameters

This section describes the provisioning parameters broadly organized according to function. It includes the following topics:

General Purpose Parameters

The general purpose parameters GPP_* are used as free string registers when configuring the IP Telephony Device to interact with a particular provisioning server solution. The GPP_* parameters are empty by default. They can be configured to contain diverse values, including the following:

- Encryption keys
- URLs
- Multistage provisioning status information
- Post request templates
- Parameter name alias maps
- Partial string values, eventually combined into complete parameter values.

The GPP_* parameters are available for macro expansion within other provisioning parameters. For this purpose, single-letter upper-case macro names (A through P) are sufficient to identify the contents of GPP_A through GPP_P. Also, the two-letter upper-case macro names SA through SD identify GPP_SA through GPP_SD as a special case when used as arguments of the **key** URL option.

For example, if GPP_A contains the string ABC, and GPP_B contains 123, the expression AB macro expands into ABC123.

Enables

All profile resync and firmware upgrade operations are controlled by the Provision_Enable and Upgrade_Enable parameters. These parameters control resyncs and upgrades independently of each other. These parameters also control resync and upgrade URL commands issued through the administration web server. Both of these parameters are set to yes by default.

In addition, the Resync_From_SIP parameter controls requests for resync operations via a SIP NOTIFY event sent from the service provider proxy server to the IP Telephony Device. If enabled, the proxy can request a resync by sending a SIP NOTIFY message containing the Event: resync header to the device.

The device challenges the request with a 401 response (authorization refused for used credentials), and expects an authenticated subsequent request before honoring the resync request from the proxy. The Event: reboot_now and Event: restart_now headers perform cold and warm restarts, respectively, are also challenged.

The two remaining enables are Resync_On_Reset and Resync_After_Upgrade_Attempt. These determine if the device performs a resync operation after power-up software reboots and after each upgrade attempt.

When enabling Resync_On_Reset, the device introduces a random delay following the boot-up sequence before actually performing the reset. The delay is a random time up to the value specified in Resync_Random_Delay (in seconds). In a pool of IP Telephony Devices, all of which are simultaneously powered up, this introduces a spread in the times at which each unit initiates a resync request to the provisioning server. This feature can be useful in a large residential deployment, in the case of a regional power failures.

Triggers

The IP Telephony Device is designed to resync with the provisioning server periodically. The resync interval is configured in Resync_Periodic (seconds). If this value is left empty, the device does not resync periodically.

The resync typically takes place when the voice lines are idle. In case a voice line is active when a resync is due, the IP Telephony Device delays the resync procedure until the line becomes idle again. However, it waits no longer than Forced_Resync_Delay (seconds). A resync may cause configuration parameter values to change. This, in turn, causes a firmware reboot, which terminates any voice connection active at the time of the resync.

If a resync operation fails because the IP Telephony Device was unable to retrieve a profile from the server, if the downloaded file is corrupt, or an internal error occurs, the device tries to resync again after a time specified in Resync_Error_Retry_Delay (seconds). If Resync_Error_Retry_Delay is set to 0, the device does not try to resync again following a failed resync attempt.

When upgrading, if an upgrade fails, a retry is performed after Upgrade_Error_Retry_Delay seconds.

Two configurable parameters are available to conditionally trigger a resync: Resync_Trigger_1 and Resync_Trigger_2. Each of these parameters can be programmed with a conditional expression (which undergoes macro expansion). If the condition in any of these parameters evaluates to true, a resync operation is triggered, as though the periodic resync timer had expired.

The following example condition triggers a resync if Line 1 failed to register for more than 5 minutes (300 seconds), and at least 10 minutes (600 seconds) have elapsed since the last resync attempt.

```
$REGTMR1 gt 300 and $PRVTMR ge 600
```

Configurable Schedules

Profile resyncs and upgrades provide for automatic retries in case of failure, in addition to periodic configuration updates. Time intervals are specified via three parameters, which are usually specified as a specific interval duration, in seconds. Starting with firmware version 3, these parameters allow the application-level (macro time scale) retry schedule to be configured. These provisioning parameters are:

- Resync_Periodic
- Resync_Error_Retry_Delay
- Upgrade_Error_Retry_Delay

These parameters accept a single delay value (seconds). The new extended syntax allows for a comma-separated list of consecutive delay elements. Each delay element consists of a deterministic delay value, optionally followed by a plus sign and an additional numeric value, which bounds a random extra delay. The last element in the sequence is implicitly repeated forever. For example,

```
Resync_Periodic            =  7200
Resync_Error_Retry_Delay   =  1800,3600,7200,14400
```

In this example, the IP Telephony Device periodically resyncs every two hours. In case of a resync failure, the device retries in 30 minutes, then again in 1 more hour, then after two more hours, and then after four more hours, continuing at four-hour intervals until it successfully resyncs.

The following is another example:

```
Resync_Periodic              =  3600+600
Resync_Error_Retry_Delay     =  1800+300,3600+600,7200+900
```

In this example, the device periodically resyncs every hour (plus an additional random delay of up to 10 minutes). In case of resync failure, the device retries in 30 minutes (plus up to five minutes more).

If it fails again, it waits an additional hour (plus up to 10 minutes). If again unsuccessful, it waits two more hours (plus up to 15 minutes), and so also thereafter, until it successfully resyncs.

The following is another example:

```
Upgrade_Error_Retry_Delay  =  1800,3600,7200,14400+3600
```

In this example, if a remote upgrade attempt fails, the device retries the upgrade in 30 minutes, then again after one more hour, then in two hours. If it still fails, it subsequently retries every four to five hours, until it succeeds.

Profile Rules

The IP Telephony Device provides multiple remote configuration profile parameters (Profile_Rule*). This means that each resync operation can retrieve multiple files, potentially managed by different servers.

In the simplest scenario, the device resyncs periodically to a single profile on a central server, which updates all pertinent internal parameters. Alternatively, the profile can be split between different files. One file is common for all the IP Telephony Devices in a deployment, while a separate file is provided that is unique for each account. Encryption keys and certificate information could be supplied by still another profile, stored on a separate server.

Whenever a resync operation is due, the IP Telephony Device evaluates the four Profile_Rule* parameters in sequence:

1. Profile_Rule
2. Profile_Rule_B

3. Profile_Rule_C
4. Profile_Rule_D

Each evaluation may result in a profile being retrieved from a remote provisioning server, possibly updating some number of internal parameters. If an evaluation fails, the resync sequence is interrupted, and is retried again from the beginning specified by the Resync_Error_Retry_Delay parameter (seconds). If all evaluations succeed, the device waits for the second specified by the Resync_Periodic parameter, and then performs a resync again.

The contents of each Profile_Rule* parameter consist of a set of alternatives. The alternatives are separated by the | (pipe) character. Each alternative consists of a conditional expression, an assignment expression, a profile URL, and any associated URL options. All these components are optional within each alternative. The following are the valid combinations, and the order in which they must appear, if present:

```
[ conditional-expr ] [ assignment-expr ] [[ options ] URL ]
```

Within each Profile_Rule* parameter, all of the alternatives except the last one must provide a conditional expression. This expression is evaluated and processed as follows:

1. Conditions are evaluated from left to right, until one is found that evaluates as true (or until one alternative is found with no conditional expression)
2. Any accompanying assignment expression is evaluated, if present
3. If a URL is specified as part of that alternative, an attempt is made to download the profile located at the specified URL, and update the internal parameters accordingly.

If all alternatives have conditional expressions, and none evaluates to true (or if the whole profile rule is empty), then the entire Profile_Rule* parameter is skipped, and the next profile rule parameter in the sequence is evaluated.

The following are some examples of valid programming for a single Profile_Rule* parameter.

The following example resyncs unconditionally to the profile at the specified URL, performing an HTTP GET request to the remote provisioning server.

```
http://remote.server.com/cisco/$MA.cfg
```

In the following example, the device resyncs to two different URLs, depending on the registration state of Line 1. In case of lost registration, the device performs an HTTP POST to a CGI script, transmitting the contents of the macro expanded GPP_A (which may provide additional information on the state of the device).

```
($REGTMR1 eq 0)? http://p.tel.com/has-reg.cfg
| [--post a] http://p.tel.com/lost-reg?
```

In the following example, the device resyncs to the same server, but provides additional information if a certificate is not installed in the unit (for legacy pre-2.0 units).

```
(“$CCERT” eq “Installed”)? https://p.tel.com/config?
| https://p.tel.com/config?cisco$MAU
```

In the following example, Line 1 is disabled until GPP_A is set equal to Provisioned through the first URL. Afterwards, it resyncs to the second URL.

```
(“$A” ne “Provisioned”)? (Line_Enable_1_ = “No”;)! https://p.tel.com/init-
prov
| https://p.tel.com/configs
```

In the following example, the profile returned by the server is assumed to contain XML element tags that need to be remapped to proper parameter names by the aliases map stored in GPP_B.

```
[--alias b] https://p.tel.com/account/spa$MA.xml
```

A resync is typically considered unsuccessful if a requested profile is not received from the server. This default behavior can be overridden by the parameter Resync_Fails_On_FNF. If Resync_Fails_On_FNF is set to No, then the device accepts a file-not-found response from the server as a successful resync. The default value for Resync_Fails_On_FNF is Yes.

Report Rule

The IP Telephony Device provides a mechanism for reporting its current internal configuration to the provisioning server. This is useful for development and debugging. The report syntax is similar to the XML profile. All provisionable parameters are included, except for the values of passwords, keys, and the GPP_SA to GPP_SD parameters, which are not shown.

The Report_Rule parameter is evaluated like a profile rule parameter. In other words, it accepts a URL, optionally qualified with a bracketed expression. The URL specifies the target destination for the report and an encryption key can be included as an option.

The URL scheme can be TFTP, HTTP, or HTTPS. When using TFTP, the operation performed is TFTP PUT. In the case of HTTP and HTTPS, the operation performed is HTTP POST.

If an encryption key is specified, the report is encrypted using 256-bit AES in CBC mode. The encrypted report can be decrypted with the following OpenSSL (or equivalent) command:

```
openssl enc –d –aes-256-cbc –k secretphrase –in rep.xml.enc –out rep.xml
```

The following is an example of the corresponding Report_Rule configuration:

```
[ --key secretphrase ] http://prov.serv.net/spa/$MA/rep.xml.enc
```

Once the report rule is configured, an actual report can be generated and transmitted by sending the device a SIP NOTIFY message, with the Event: report type. The SIP NOTIFY request is handled like other SIP notifies, with the device requiring authentication from the requesting server before honoring the request to issue a report. Each SIP NOTIFY report request generates one attempt to transmit the report. Retries are not supported.

Upgrade Rule

The IP Telephony Device provides one configurable remote upgrade parameter, Upgrade_Rule. This parameter accepts a syntax similar to the profile rule parameters. URL options not supported for upgrades, but conditional expressions and assignment expressions can be used. If conditional expressions are used, the parameter can be populated with multiple alternatives, separated by the | character. The syntax for each alternative is as follows:

```
[ conditional-expr ] [ assignment-expr ] URL
```

As in the case of Profile_Rule* parameters, the Upgrade_Rule parameter evaluates each alternative until a conditional expression is satisfied or an alternative has no conditional expression. The accompanying assignment expression is evaluated, if specified. Then, an upgrade to the specified URL is attempted.

If the Upgrade_Rule contains a URL without a conditional expression, the device upgrades to the firmware image specified by the URL. Subsequently, it does not attempt to upgrade again until either the rule itself is modified or the effective combination of scheme + server + port + filepath is changed, following macro expansion and evaluation of the rule.

In order to attempt a firmware upgrade, the device disables audio at the start of the procedure, and reboots at the end of the procedure. For this reason, an upgrade driven by the contents of Upgrade_Rule is only automatically initiated by the device if any voice line is currently inactive.

For example,

```
http://p.tel.com/firmware/spa021025.bin
```

In this example, the Upgrade_Rule upgrades the firmware to the image stored at the indicated URL. The following is another example:

```
(“$F” ne “beta-customer”)? http://p.tel.com/firmware/spa021025.bin
| http://p.tel.com/firmware/spa-test-0527s.bin
```

This example directs the unit to load one of two images, based on the contents of a general purpose parameter, GPP_F.

The device can enforce a downgrade limit with respect to firmware revision number. This can be useful as a customization option. If a valid firmware revision number is configured in the parameter Downgrade_Rev_Limit, the device rejects upgrade attempts for firmware versions earlier than the specified limit.

Data Types

The data types used with configuration profile parameters are as follows:

- Uns<n>—Unsigned n-bit value, where n = 8, 16, or 32. It can be specified in decimal or hex format such as 12 or 0x18 as long as the value can fit into n bits.
- Sig<n>—Signed n-bit value. It can be specified in decimal or hex format. Negative values must be preceded by a “-“ sign. A + sign before positive value is optional.
- Str<n>—A generic string with up to n non-reserved characters.
- Float<n>—A floating point value with up to n decimal places.

- Time<n>—Time duration in seconds, with up to n decimal places. Extra decimal places specified are ignored.
- PwrLevel—Power level expressed in dBm with 1 decimal place, such as –13.5 or 1.5 (dBm).
- Bool—Boolean value of either “yes” or “no.”
- {a,b,c,...}—A choice among a, b, c, ...
- IP—IP Address in the form of x.x.x.x, where x between 0 and 255. For example 10.1.2.100.
- Port—TCP/UDP Port number (0-65535). It can be specified in decimal of hex format.
- UserID—User ID as appeared in a URL; up to 63 characters.
- FQDN—Fully Qualified Domain Name, such as “sip.Cisco.com:5060”, or “109.12.14.12:12345”. It can contain up to 63 characters.
- Phone—A phone number string, such as 14081234567, *69, *72, 345678, or a generic URL such as 1234@10.10.10.100:5068, or jsmith@Cisco.com. It can contain up to 39 characters.
- ActCode—Activation code for a supplementary service, such as *69. It can contain up to 7 characters.
- PhTmplt—A phone number template. Each template may contain one or more patterns separated by a “,”. White space at the beginning of each pattern is ignored. “?” and “*” represent wildcard characters. To represent literally use %xx. For example, %2a represents *. It can contain up to 39 characters. Examples: “1408*, 1510*”, “14081234????, 555?1.”.
- RscTmplt—A template of SIP Response Status Code, such as “404, 5*”, “61?”, “407, 408, 487, 481”. It can contain up to 39 characters.
- CadScript—A mini-script that specifies the cadence parameters of a signal. Up to 127 characters. Syntax: $S_1[;S_2]$, where: $S_i=D_i(on_{i,1}/off_{i,1}[,on_{i,2}/off_{i,2}[,on_{i,3}/off_{i,3}[,on_{i,4}/off_{i,4}[,on_{i,5}/off_{i,5}[,on_{i,6}/off_{i,6}]]]]])$ and is known as a *section*, $on_{i,j}$ and $off_{i,j}$ are the on/off duration in seconds of a *segment* and i = 1 or 2, and j = 1 to 6. D_i is the total duration of the section in seconds. All durations can have up to three decimal places to provide 1 ms resolution. The wildcard character “*” stands for infinite duration. The segments within a section are played in order and repeated until the total duration is played.

Example 1:

```
60(2/4)

Number of Cadence Sections = 1
Cadence Section 1: Section Length = 60 s
Number of Segments = 1
Segment 1: On=2s, Off=4s

Total Ring Length = 60s
```

Example 2—Distinctive ring (short,short,short,long):

```
60(.2/.2,.2/.2,.2/.2,1/4)

Number of Cadence Sections = 1
Cadence Section 1: Section Length = 60s
Number of Segments = 4
Segment 1: On=0.2s, Off=0.2s
Segment 2: On=0.2s, Off=0.2s
Segment 3: On=0.2s, Off=0.2s
Segment 4: On=1.0s, Off=4.0s

Total Ring Length = 60s
```

- FreqScript—A mini-script that specifics the frequency and level parameters of a tone. Up to 127 characters. Syntax: $F_1@L_1[,F_2@_{L2}[,F_3@L_3[,F_4@L_4[,F_5@L_5[,F_6@L_6]]]]]$, where F_1–F_6 are frequency in Hz (unsigned integers only) and L_1–L_6 are corresponding levels in dBm (with up to 1 decimal places). White spaces before and after the comma are allowed (but not recommended).

  Example 1—Call Waiting Tone:

  ```
  440@-10
  ```

```
Number of Frequencies = 1
   Frequency 2 = 440 Hz at –10 dBm
```

Example 2—Dial Tone:

```
350@-19,440@-19

Number of Frequencies = 2
   Frequency 1 = 350 Hz at –19 dBm
   Frequency 2 = 440 Hz at –19 dBm
```

- ToneScript—A mini-script that specifies the frequency, level and cadence parameters of a call progress tone. May contain up to 127 characters. Syntax: FreqScript;Z_1[;Z_2]. The section Z_1 is similar to the S_1 section in a CadScript except that each on/off segment is followed by a frequency components parameter: $Z_1 = D_1(on_{i,1}/off_{i,1}/f_{i,1}[,on_{i,2}/off_{i,2}/f_{i,2}\ [,on_{i,3}/off_{i,3}/f_{i,3}\ [,on_{i,4}/off_{i,4}/f_{i,4}\ [,on_{i,5}/off_{i,5}/f_{i,5}\ [,on_{i,6}/off_{i,6}/f_{i,6}]]]]])$, where $fi,j = n_1[+n_2]+n_3[+n_4[+n_5[+n_6]]]]$ and $1 < n_k < 6$ indicates which of the frequency components given in the FreqScript are used in that segment; if more than one frequency component is used in a segment, the components are summed together.

  Example 1—Dial tone:

```
350@-19,440@-19;10(*/0/1+2)

Number of Frequencies = 2
Frequency 1 = 350 Hz at –19 dBm
Frequency 2 = 440 Hz at –19 dBm
Number of Cadence Sections = 1
Cadence Section 1: Section Length = 10 s
Number of Segments = 1
Segment 1: On=forever, with Frequencies 1 and 2

Total Tone Length = 10s
```

Example 2—Stutter tone:

```
350@-19,440@-19;2(.1/.1/1+2);10(*/0/1+2)

Number of Frequencies = 2
Frequency 1 = 350 Hz at –19 dBm
Frequency 2 = 440 Hz at –19 dBm
Number of Cadence Sections = 2
Cadence Section 1: Section Length = 2s
Number of Segments = 1
Segment 1: On=0.1s, Off=0.1s with Frequencies 1 and 2
Cadence Section 2: Section Length = 10s
Number of Segments = 1
Segment 1: On=forever, with Frequencies 1 and 2

Total Tone Length = 12s
```

Example 3—SIT tone:

```
985@-16,1428@-16,1777@-16;20(.380/0/1,.380/0/2,.380/0/
3,0/4/0)

Number of Frequencies = 3
Frequency 1 = 985 Hz at –16 dBm
Frequency 2 = 1428 Hz at –16 dBm
Frequency 3 = 1777 Hz at –16 dBm
Number of Cadence Sections = 1
Cadence Section 1: Section Length = 20s
Number of Segments = 4
Segment 1: On=0.38s, Off=0s, with Frequency 1
Segment 2: On=0.38s, Off=0s, with Frequency 2
Segment 3: On=0.38s, Off=0s, with Frequency 3
Segment 4: On=0s, Off=4s, with no frequency components

Total Tone Length = 20s
```

- ProvisioningRuleSyntax—Scripting syntax used to define configuration resync and firmware upgrade rules.
- DialPlanScript—Scripting syntax used to specify Line 1 and Line 2 dial plans.

NOTE

- `<Par Name>` represents a configuration parameter name. In a profile, the corresponding tag is formed by replacing the space with an underscore "_", such as **Par_Name**.
- An empty default value field implies an empty string < "" >.
- The IP Telephony Device continues to use the last configured values for tags that are not present in a given profile.
- Templates are compared in the order given. The first, *not the closest*, match is selected. The parameter name must match exactly.
- If more than one definition for a parameter is given in a configuration file, the last such definition in the file is the one that takes effect in the IP Telephony Device.
- A parameter specification with an empty parameter value forces the parameter back to its default value. To specify an empty string instead, use the empty string "" as the parameter value.

3

Provisioning Tutorial

This chapter describes the procedures for transferring configuration profiles between the IP Telephony Device and the provisioning server and includes the following sections:

For information about creating configuration profiles, refer to **Chapter 2, "Creating Provisioning Scripts."**

Preparation

The examples presented in this chapter require the availability of one or more servers. For the purposes of this tutorial, these can be installed and run on a local PC. To troubleshoot server configuration, it is helpful to install clients for each type of server on a separate server machine. That establishes proper server operation independent of the interaction with Cisco Small Business VoIP devices.

The pertinent servers include: syslog (UDP port 514), TFTP (UDP port 69), HTTP (TCP port 80), HTTPS (TCP port 443). For generating configuration profiles, it is useful to install the open source gzip compression utility. For profile encryption and HTTPS operations, you can install the open source OpenSSL software package. In addition, to test dynamic generation of profiles and one-step remote provisioning using HTTPS, a scripting language with CGI scripting support, such as the open source Perl language tools, is recommended.

Finally, to verify secure exchanges between provisioning servers and Cisco Small Business voice devices, it is useful to install an Ethernet packet sniffer (such as the freely downloadable Ethereal/Wireshark). For HTTPS transactions, you can use the ssldump utility.

An IP Telephony Device can retrieve a configuration profile from a provisioning server and update its internal configuration accordingly. IP Telephony Devices accept two different profile formats, one based on an open published syntax, and one based on an unpublished binary definition. The open configuration profile format uses a simple XML-like syntax. The binary format is generated by converting a plain text file using the SIP Profiler Compiler (SPC).

The examples in this tutorial use configuration profiles with XML-style syntax. To use the proprietary plain-text format, you need to convert the files using SPC before they can be used. This procedure is described in the **"Proprietary Profile Format" section on page 81**.

Basic Resync

This section demonstrates the basic resync functionality of Cisco Small Business VoIP devices. It includes the following topics:

TFTP Resync

The IP Telephony Device supports multiple network protocols for retrieving configuration profiles. The most basic profile transfer protocol is TFTP (RFC1350). TFTP is widely used for the provisioning of network devices within private LAN networks. Although not recommended for deployments of endpoints across the Internet, it can be convenient for deployment within small organizations, for in-house preprovisioning, and for development and testing.

The following configuration profile format uses the XML-style syntax:

```
<flat-profile>
  <GPP_A> 12345678
  </GPP_A>
</flat-profile>
```

The `<flat-profile>` element tag encloses all parameter elements to be recognized by the IP Telephony Device. The example above defines one parameter value, the first general purpose parameter (GPP_A), with a value of 12345678.

Exercise

STEP 1 Within a LAN environment connect a PC and an IP Telephony Device to a hub, switch, or small router.

STEP 2 Connect an analog phone to the Phone 1 port of the IP Telephony Device.

STEP 3 On the PC, install and activate a TFTP server.

STEP 4 Using a text editor, create the configuration profile and save it with the name `basic.txt` in the virtual root directory of the installed TFTP server.

STEP 5 If possible, verify that the TFTP server is properly configured by requesting the basic.txt file using a TFTP client other than the IP Telephony Device itself.

Preferably, the TFTP client should be running on a separate host from the server.

STEP 6 Using the analog phone, obtain the IP address of the IP Telephony Device (IVR menu ****** 110 #**).

If the configuration has been modified since it was manufactured, perform factory reset on it by using the IVR RESET option (******** 73738#).

STEP 7 Open the PC web browser on the admin/advanced configuration page.

For example, if the IP address is 192.168.1.100):

```
http://192.168.1.100/admin/advanced
```

STEP 8 The Provisioning tab in the admin/advanced page contains a number of configurable parameters specific to provisioning. Select the Provisioning tab, and inspect the values of the general purpose parameters GPP_A through GPP_P.

These should be empty.

STEP 9 To resync the test IP Telephony Device to the `basic.txt` configuration profile, open the following URL from the PC browser.

Assuming the PC IP address is 192.168.1.200:

```
http://192.168.1.100/admin/resync?tftp://192.168.1.200/basic.txt
```

This resync URL method is designed for development and testing. When it receives this command, the IP Telephony Device at address 192.168.1.100 requests the file `basic.txt` from the TFTP server at IP address 192.168.1.200. It then parses the downloaded file and updates the GPP_A parameter with the value 12345678.

STEP 10 Verify that the parameter was correctly updated by refreshing the admin/advanced page on the PC web browser and selecting the Provisioning tab on that page.

The GPP_A parameter should now contain the value 12345678.

Logging with syslog

The IP Telephony Device sends a syslog message to the designated syslog server when the device is about to resync to a provisioning server and after the resync has either completed or failed. This server is identified in the web server administration (admin/advanced, System tab, Syslog_Server parameter). It is instructive to configure the syslog server IP address into the device and observe the messages generated during each exercise.

Exercise

STEP 1 Install and activate a syslog server on the local PC.

STEP 2 Program the PC IP address into the Syslog_Server parameter, and submit the change.

Click the **System** tab and enter the value of your local syslog server into the Syslog_Server parameter.

STEP 3 Repeat the TFTP Resync operation described in the previous exercise.

The device generates two syslog messages during the resync. The first indicates that a request is in progress. The second marks success or failure of the resync.

STEP 4 Verify that your syslog server received messages such as the following:

```
SPA-962 00:0e:08:ab:cd:ef -- Requesting resync tftp://192.168.1.200/basic.txt
SPA-962 00:0e:08:ab:cd:ef -- Successful resync tftp://192.168.1.200/basic.txt
```

More detailed messages are available by programming the Debug_Server parameter (instead of the Syslog_Server parameter) with the IP address of the syslog server, and setting the Debug_Level to a value between 0 and 3 (3 being the most verbose).

The contents of these messages can be configured using the following parameters:

- Log_Resync_Request_Msg
- Log_Resync_Success_Msg
- Log_Resync_Failure_Msg.

If any of these parameters are cleared, the corresponding syslog message is not generated.

Occasionally, it may also be informative to capture an Ethernet packet trace of the interaction between the IP Telephony Device and the provisioning server. You can run the Ethernet packet analyzer (such as Ethereal/Wireshark) on a PC that is connected, through a hub or through a switch with port mirroring enabled, to the same subnet as the IP Telephony Device.

Automatic Resync

When a IP Telephony Device is deployed remotely or as part of an internal company deployment, it may need to resync periodically to the provisioning server, to ensure that any customer profile configuration changes made on the server are propagated to the endpoint, without requiring an explicit resync request to the endpoint.

To cause the IP Telephony Device to automatically and periodically resync to a server, a configuration profile URL is defined using the Profile_Rule parameter, and a resync period is defined using the Resync_Periodic parameter.

Exercise

STEP 1 Using the PC web browser, open the admin/advanced page, Provisioning tab.

STEP 2 Define the Profile_Rule parameter.

STEP 3 The following value assumes a TFTP server IP address of 192.168.1.200:

```
tftp://192.168.1.200/basic.txt
```

STEP 4 In the Resync_Periodic parameter enter a small value for testing such as **30** (meaning 30 seconds).

STEP 5 Click **Submit all Changes**.

With the new parameter settings, the IP Telephony Device now resyncs to the configuration file specified by the URL twice a minute.

STEP 6 Observe the resulting messages in the syslog trace.

STEP 7 Ensure that the Resync_On_Reset parameter is set to **yes**.

STEP 8 Power cycle the IP Telephony Device.

The IP Telephony Device also automatically resyncs to the provisioning server whenever it is power-cycled.

If the resync operation fails for any reason, such as if the server is not responding, the unit waits the number of seconds defined in Resync_Error_Retry_Delay before attempting to resync again. If Resync_Error_Retry_Delay is zero, the IP Telephony Device does not try to resync following a failed resync attempt.

STEP 9 (Optional) Verify that the value of Resync_Error_Retry_Delay is set to a small number, such as **30**, disable the TFTP server, and observe the results in the syslog output.

Unique Profiles and Macro Expansion

In a large deployment, each IP Telephony Device needs to be configured with distinct values for specific parameters, such as User_ID or Display_Name. To meet this requirement, the service provider must generate distinct profiles, one for each deployed device. Each IP Telephony Device, in turn, must be configured to resync to its own profile, according to some predetermined profile naming convention.

The profile URL syntax can include identifying information specific to each IP Telephony Device (such as MAC address and serial number) via macro expansion of built-in variables. This eliminates the need to specify these values within each profile.

The profile rule undergoes macro expansion internally before being applied. The macro expansion understands a number of values including the following:

- $MA expands to the unit MAC address, using lower case hex digits (for example, 000e08abcdef)
- $SN expands to the unit Serial Number (for example, 88012BA01234)

Exercise

STEP 1 Obtain the MAC address of the test IP Telephony Device from its product label.

This is a 12-digit number, using lower case hex digits, such as 000e08aabbcc.

STEP 2 Copy the basic.txt configuration file to a new file named spa_*mac_address*.cfg and place the new file in the virtual root directory of the TFTP server.

Replace *mac_address* with the actual MAC address of the IP Telephony Device.

STEP 3 Open the admin/advanced page, Provisioning tab.

STEP 4 Enter the following value in the Profile_Rule parameter:
tftp://192.168.1.200/spa$MA.cfg

STEP 5 Click **Submit All Changes**.

This causes an immediate reboot and resync.

When the next resync occurs, the IP Telephony Device retrieves the new file by expanding the $MA macro expression into its own MAC address.

Several other values can be macro expanded in this way, including all the general purpose parameters, (GPP_A through GPP_P) These can be referenced as $A through $P. Macro expansion is not limited to the URL file name, but can also be applied to any portion of the profile rule parameter.

For a complete list of variables available for macro expansion, see the **"Macro Expansion Variables" section on page 91**.

URL Resolution

The profile URL can contain a provisioning server name instead of an explicit IP address. In this case, the IP Telephony Device performs a DNS lookup to resolve the name.

A non-standard server port can be specified in the URL, using the standard syntax :port following the server name.

Also, the configuration profile can be stored in a subdirectory of the server virtual root directory. Again, this is specified using standard URL notation.

For example, the following is a valid Profile_Rule that requests the file spa962.cfg, in the server subdirectory /cisco/config, for the TFTP server running on host prov.telco.com, which listens for connection on port 6900.

```
tftp://prov.telco.com:6900/cisco/config/spa962.cfg
```

Again, macro expansion can be used anywhere in the URL. This can be convenient in organizing a directory of profiles on the server for the deployed devices. For example, a profile subdirectory name might be supplied for each IP Telephony Device in a dedicated general purpose parameter, with its value referred within a common profile rule via macro expansion.

For example, GPP_B has the following definition:

```
Dj6Lmp23Q
```

The Profile_Rule has this value:

```
tftp://prov.telco.com/cisco/$B/$MA.cfg
```

Then, when resyncing, this IP Telephony Device (assuming a MAC address of 000e08012345) requests the profile at the following URL:

```
tftp://prov.telco.com/cisco/Dj6Lmp23Q/000e08012345.cfg
```

HTTP GET Resync

HTTP provides a more reliable resync mechanism than TFTP because HTTP establishes a TCP connection and TFTP uses UDP, which is less reliable. In addition, HTTP servers offer improved filtering and logging features compared to TFTP servers.

On the client side, using HTTP (with the GET method) simply means changing TFTP to HTTP in the URL defined in the Profile_Rule parameter.

On the server side, the service provider must install and configure the HTTP server. The IP Telephony Device does not require any special configuration setting on the server to be able to resync using HTTP. If a standard web browser can retrieve a profile from a particular server using HTTP, the IP Telephony Device should be able to do so as well.

Exercise

STEP 1 Install an HTTP server on the local PC or other accessible host.

The open source Apache server can be downloaded from the Internet.

STEP 2 Copy the basic.txt configuration profile from the earlier exercises onto the virtual root directory of the installed server.

STEP 3 Verify proper server installation (and file access of basic.txt) by accessing the profile using a standard web browser.

STEP 4 Modify the Profile_Rule of the test IP Telephony Device to point to the HTTP server in place of the TFTP server, so as to download its profile periodically.

For example, assuming the HTTP server is at 192.168.1.300, enter the following value:

```
http://192.168.1.200/basic.txt
```

STEP 5 Observe the syslog messages sent by the IP Telephony Device.

The periodic resyncs should now be obtaining the profile from the HTTP server.

Also, the server should be logging each request if connection logging is enabled in the server configuration.

STEP 6 In the HTTP server logs, observe how information identifying the test IP Telephony Device appears in the log of user agents.

This should include the manufacturer, product name, current firmware version, and serial number.

Secure Resync

This section demonstrates the preferred mechanisms available on the IP Telephony Device for securing the provisioning process. It includes the following topics:

- **Basic HTTPS Resync, page 72**
- **HTTPS With Client Certificate Authentication, page 74**
- **HTTPS Client Filtering and Dynamic Content, page 75**

Basic HTTPS Resync

HTTPS adds SSL to HTTP for remote provisioning so that:

- The IP Telephony Device can authenticate the provisioning server.
- The provisioning server can authenticate the IP Telephony Device.
- The confidentiality of information exchanged between the IP Telephony Device and the provisioning server is ensured through encryption.

SSL generates and exchanges secret (symmetric) keys for each connection between the IP Telephony Device and the server, using public/private key pairs preinstalled in the IP Telephony Device and the provisioning server.

On the client side, using HTTPS (with the GET method), simply requires changing the definition of the URL in the Profile_Rule parameter from **http** to **https**. On the server side, the service provider must install and set up the HTTPS server.

In addition, an SSL server certificate signed by Cisco must be installed on the provisioning server. The devices cannot resync to a server using HTTPS, unless the server supplies a Cisco-signed server certificate.

Exercise

STEP 1 Install an HTTPS server on a host whose IP address is known to the network DNS server, through normal hostname translation.

The open source Apache server can be configured to operate as an HTTPS server, when installed with the open source mod_ssl package.

STEP 2 Generate a server Certificate Signing Request for the server.

STEP 3 For this step, you may need to install the open source OpenSSL package or equivalent software. If using OpenSSL, the command to generate the basic CSR file is as follows:

```
openssl req –new –out provserver.csr
```

This command generates a public/private key pair, which is saved in the privkey.pem file.

STEP 4 Submit the CSR file (provserver.csr) to Cisco for signing.

A signed server certificate is returned (provserver.cert) along with a Sipura CA Client Root Certificate, spacroot.cert.

STEP 5 Store the signed server certificate, the private key pair file, and the client root certificate in the appropriate locations on the server.

In the case of an Apache installation on Linux, these locations are typically as follows:

```
# Server Certificate:
SSLCertificateFile /etc/httpd/conf/provserver.cert
# Server Private Key:
SSLCertificateKeyFile /etc/httpd/conf/pivkey.pem
# Certificate Authority:
SSLCACertificateFile /etc/httpd/conf/spacroot.cert
```

STEP 6 Restart the server.

STEP 7 Copy the basic.txt configuration profile from the earlier exercises onto the virtual root directory of the HTTPS server.

STEP 8 Verify proper server operation by downloading basic.txt from the HTTPS server, using a standard browser from the local PC.

STEP 9 Inspect the server certificate supplied by the server.

The browser probably does not recognize it as valid unless the browser has been preconfigured to accept Cisco as a root CA. However, the IP Telephony Devices expect the certificate to be signed this way.

Modify the Profile_Rule of the test device to contain a reference to the HTTPS server in place of the HTTP server, for example:

```
https://my.server.com/basic.txt
```

This example assumes the name of the HTTPS server is my.server.com.

STEP 10 Click **Submit All Changes**.

STEP 11 Observe the syslog trace sent by the IP Telephony Device.

The syslog message should indicate that the resync obtained the profile from the HTTPS server.

STEP 12 (Optional) Use an Ethernet protocol analyzer on the IP Telephony Device subnet to verify that the packets are encrypted.

STEP 13 In this exercise, client certificate verification is not yet enabled, use a browser to request the profile stored in basic.txt.

At this point, the connection between IP Telephony Device and server is encrypted. However, the transfer is not secure because any client can connect to the server and request the file, given knowledge of the file name and directory location. For secure resync, the server must also authenticate the client, as demonstrated in the next exercise.

HTTPS With Client Certificate Authentication

In the factory default configuration, the server does not request an SSL client certificate from a client. After you edit the configuration to enable client authentication, then the server requires a client certificate to authenticate the IP Telephony Device before accepting a connection request.

Because of this, the resync operation in this exercise cannot be independently tested using a browser lacking the proper credentials. Nevertheless, the SSL key exchange within the HTTPS connection between the test IP Telephony Device and the server can be observed using the ssldump utility. The utility trace shows the interaction between client and server.

Exercise

STEP 1 Enable client certificate authentication on the HTTPS server.

STEP 2 In Apache (v.2), set the following in the server configuration file:

```
SSLVerifyClient  require
```

Also ensure that the spacroot.cert has been stored as shown in the previous exercise.

STEP 3 Restart the HTTPS server and observe the syslog trace from the IP Telephony Device.

Each resync to the server now performs symmetric authentication, so that both the server certificate and the client certificate are verified before the profile is transferred.

STEP 4 Using ssldump, capture a resync connection between the IP Telephony Device and the HTTPS server.

If client certificate verification is properly enabled on the server, the ssldump trace shows the symmetric exchange of certificates (first server-to-client, then client-to-server) before the encrypted packets containing the profile.

With client authentication enabled, only a IP Telephony Device with a MAC address matching a valid client certificate can request the profile from the provisioning server. A request from an ordinary browser or other unauthorized device is rejected by the server.

HTTPS Client Filtering and Dynamic Content

If the HTTPS server is configured to require a client certificate, then the information in the certificate identifies the resyncing IP Telephony Device and supplies it with the correct configuration information.

The HTTPS server makes the certificate information available to CGI scripts (or compiled CGI programs) invoked as part of the resync request. For the purpose of illustration, this exercise uses the open source Perl scripting language, and assumes that Apache (v.2) is used as the HTTPS server.

Exercise

STEP 1 Install Perl on the host running the HTTPS server.

STEP 2 Generate the following Perl reflector script:

```
#!/usr/bin/perl -wT
use strict;
print “Content-Type: text/plain\n\n”;
print “<flat-profile><GPP_D>”;

print “OU=$ENV{‘SSL_CLIENT_I_DN_OU’},\n”;
print “L=$ENV{‘SSL_CLIENT_I_DN_L’},\n”;
print “S=$ENV{‘SSL_CLIENT_I_DN_S’}\n”;

print “</GPP_D></flat-profile>”;
```

STEP 3 Save this file with the file name reflect.pl, with executable permission (chmod 755 on Linux), in the CGI scripts directory of the HTTPS server.

STEP 4 Verify accessibility of CGI scripts on the server (as in /cgi-bin/...).

STEP 5 Modify the Profile_Rule on the test device to resync to the reflector script, as in the following example:

```
https://prov.server.com/cgi-bin/reflect.pl?
```

STEP 6 Click **Submit All Changes**.

STEP 7 Observe the syslog trace to ensure a successful resync.

STEP 8 Open the admin/advanced page, Provisioning tab.

STEP 9 Verify that the GPP_D parameter contains the information captured by the script.

This information contains the product name, MAC address, and serial number if the test device carries a unique certificate from the manufacturer, or else generic strings if it is a unit manufactured before firmware release 2.0.

A similar script could be used to determine information about the resyncing device and then provide it with appropriate configuration parameter values.

Profile Formats

This section demonstrates the generation of configuration profiles. To explain the functionality in this section, TFTP from a local PC is used as the resync method, although HTTP or HTTPS can be used for testing as well, if it is convenient. This section includes the following topics:

Profile Compression

A configuration profile in XML format can become quite large if all parameters are individually specified by the profile. To reduce the load on the provisioning server, the IP Telephony Device supports compression of the XML file, using the deflate compression format used by the gzip utility (RFC 1951).

Exercise

STEP 1 Install gzip on the local PC.

STEP 2 Compress the basic.txt profile from earlier exercises, by invoking gzip from the command line:

```
gzip basic.txt
```

This generates the deflated file basic.txt.gz.

STEP 3 Save the deflated file in the TFTP server virtual root directory.

STEP 4 Modify the Profile_Rule on the test device to resync to the deflated file in place of the original XML file, as in the following example:

```
tftp://192.168.1.200/basic.txt.gz
```

STEP 5 Click **Submit All Changes**.

STEP 6 Observe the syslog trace from the IP Telephony Device.

Upon resync, the new file is downloaded by the IP Telephony Device and used to update its parameters.

The file size of such a small profile is not reduced by gzip. Compression is only useful with larger profiles.

For integration into customized back-end provisioning server solutions, the open source zlib compression library can be used in place of the standalone gzip utility to perform the profile compression. However, the IP Telephony Device expects the file to contain a valid gzip header.

Profile Encryption

A compressed or uncompressed profile can be encrypted. This is useful when the confidentiality of the profile information is of particular concern, such as when using TFTP or HTTP for communication between the IP Telephony Device and the provisioning server.

The IP Telephony Device supports symmetric key encryption using the 256-bit AES algorithm. This encryption can be performed using the open source OpenSSL package.

Exercise

STEP 1 Install OpenSSL on a local PC.

This may require recompilation to enable the AES code.

STEP 2 Starting from the XML profile in basic.txt, generate an encrypted file with the following command:

```
openssl enc –aes-256-cbc –k MyOwnSecret –in basic.txt –out basic.cfg
```

The compressed basic.txt.gz file could be used instead because the XML profile can be both compressed and encrypted.

STEP 3 Store the encrypted file basic.cfg in the TFTP server virtual root directory.

STEP 4 Modify the Profile_Rule on the test device to resync to the encrypted file in place of the original XML file. The encryption key is made known to the IP Telephony Device with the following URL option:

```
[--key MyOwnSecret ] tftp://192.168.1.200/basic.cfg
```

STEP 5 Click **Submit All Changes**.

STEP 6 Observe the syslog trace from the IP Telephony Device.

On resync, the new file is downloaded by the IP Telephony Device and used to update its parameters.

Partitioned Profiles

An IP Telephony Device downloads multiple separate profiles during each resync. This allows managing different kinds of profile information on separate servers and maintaining common configuration parameter values separate from account specific values.

Exercise

STEP 1 Create a new XML profile, basic2.txt, that specifies a value for a distinct parameter from the earlier exercises.

For instance, the file can contain the following:

```
<flat-profile><GPP_B>ABCD</GPP_B></flat-profile>
```

STEP 2 Store the basic2.txt profile in the virtual root directory of the TFTP server.

STEP 3 Leave the first profile rule as in any the earlier exercises, but configure the second profile rule (Profile_Rule_B) to point to the new file:

```
tftp://192.168.1.200/basic2.txt
```

STEP 4 Click **Submit All Changes**.

The IP Telephony Device now resyncs to both the first and second profiles, in that order, whenever a resync operation is due.

STEP 5 Observe the syslog trace to confirm the expected behavior.

Parameter Name Aliases

When generating an XML profile for the IP Telephony Device, it may be convenient to assign names to certain configuration parameters that are different from the canonical names recognized by the IP Telephony Device. For example, a customer account database may generate XML element tags for a customer telephone number and SIP registration password with names such as SIP-number and SIP-password. These names can be mapped to the canonical names (User_ID_1_ and Password_1_) before being applied to Line 1.

In many instances, the back-end provisioning solution used by the service provider can perform this mapping. However, the IP Telephony Device itself can remap the parameter names internally. To do this, an alias map is defined and stored in one of the general purpose provisioning parameters. Then, the profile rule which invokes the resync is directed to remap the non-canonical XML elements as specified by the alias map.

Exercise

STEP 1 Generate a profile named customer.XML containing the proprietary customer-account XML form indicated in the following example:

```
<customer-account>
  <SIP-number> 17775551234
  </SIP-number>
  <SIP-password> 512835907884
  </SIP-password>
</customer-account>
```

STEP 2 Store the file in the TFTP server virtual root directory.

STEP 3 Open the test web interface on the admin/advanced page, Provisioning tab, and edit GPP_A to contain the alias map indicated above (do not enter new lines through the web interface, instead simply enter each alias consecutively).

```
/customer-account/SIP-number = /flat-profile/User_ID_1_ ;
/customer-account/SIP-password = /flat-profile/Password_1_ ;
```

STEP 4 Edit the Profile_Rule to point to the new XML profile, and also specify the alias map as a URL option, as follows:

```
[--alias a ] tftp://192.168.1.200/customer.xml
```

STEP 5 Click **Submit All Changes**.

When the IP Telephony Device resyncs, it receives the XML profile, remaps the elements, as indicated by the alias map, and populates the User_ID_1_ and Password_1_ parameters.

STEP 6 View the Line 1 tab to verify the new configuration.

NOTE The IP Telephony Device supports alias remapping of a limited number of parameters. It is not meant to rename all parameters in its configuration.

Proprietary Profile Format

Firmware releases prior to 2.0.6 do not recognize the XML-based profiles described so far in this chapter. Instead, the SIP Profiler Compiler tool (SPC) converts a text-based profile definition into a proprietary binary format understood by earlier versions of the firmware. The tool provides its own options for encrypting the resulting binary profile.

The text-based profile understood by SPC uses a different syntax from the XML profile presented earlier. It consists of a list of parameter-value pairs, with the value in double quotes. Other minor syntax and parameter naming differences also apply. The following example specifies values for two Line 1 parameters:.

Exercise

STEP 1 Obtain the SPC utility from Cisco.

Executables are available for the Windows Win32 environment, Linux ELF, and OpenBSD.

STEP 2 Generate the text profile account.txt containing the two-line profile shown in the following example:

```
User_ID[1] “17775551234” ;
Password[1] “512835907884” ;
```

STEP 3 Compile the text profile into a binary file, account.cfg, using the following command:

```
spc account.txt account.cfg
```

STEP 4 Store account.cfg in the TFTP server virtual root directory.

STEP 5 Modify the test profile rule to point to the new profile:

```
tftp://192.168.1.200/account.cfg
```

STEP 6 Click **Submit All Changes**.

Upon resync, the IP Telephony Device retrieves the new file, recognizes its binary format and updates the two specified parameters.

STEP 7 Observe the syslog messages sent by the IP Telephony Device during resync.

4

Provisioning Field Reference

This chapter provides a listing of the parameters provided on the administration web server Provisioning tab, which can be used in configuration profile scripts. It includes the following sections:

The Provisioning parameters described in this chapter are recognized by the IP Telephony Devices beginning with firmware release 2.0.6.

Configuration Profile Parameters

The following table defines the function and usage of each parameter in the Configuration Profile Parameters section under the Provisioning tab.

Parameter Name	Description and Default Value
Provision_Enable	Controls all resync actions independently of firmware upgrade actions. Set to yes to enable remote provisioning. The default value is Yes.
Resync_On_Reset	Triggers a resync after every reboot except for reboots caused by parameter updates and firmware upgrades. The default value is Yes.
Resync_Random_Delay	The maximum value for a random time interval that the device waits before making its initial contact with the provisioning server. This delay is effective only on the initial configuration attempt following device power-on or reset. The delay is a pseudo-random number between zero and this value. This parameter is in units of 20 seconds; the default value of 2 represents 40 seconds. This feature is disabled when this parameter is set to zero. This feature can be used to prevent an overload of the provisioning server when a large number of devices power-on simultaneously. The default value is 2 (40 seconds).

Parameter Name	Description and Default Value
Resync_Periodic	The time interval between periodic resyncs with the provisioning server. The associated resync timer is active only after the first successful sync with the server. Set this parameter to zero to disable periodic resyncing. The default value is 3600 seconds.
Resync_Error_Retry_Delay	Resync retry interval (in seconds) applied in case of resync failure. The device has an error retry timer that activates if the previous attempt to sync with the provisioning server fails. The device waits to contact the server again until the timer counts down to zero. This parameter is the value that is initially loaded into the error retry timer. If this parameter is set to zero, the device immediately retries to sync with the provisioning server following a failed attempt. The default value is 3600 seconds.
Forced_Resync_Delay	Maximum delay (in seconds) the IP Telephony Device waits before performing a resync. The device does not resync while one of its phone lines is active. Because a resync can take several seconds, it is desirable to wait until the device has been idle for an extended period before resyncing. This allows a user to make calls in succession without interruption. The device has a timer that begins counting down when all of its lines become idle. This parameter is the initial value of the counter. Resync events are delayed until this counter decrements to zero. The default value is 14,400 seconds.

Parameter Name	Description and Default Value
Resync_From_SIP	Enables a resync to be triggered via a SIP NOTIFY message. The default value is Yes.
Resync_After_Upgrade_Attempt	Triggers a resync after every firmware upgrade attempt. The default value is Yes.
Resync_Trigger_1, Resync_Trigger_2	Configurable resync trigger conditions. A resync is triggered when the logic equation in these parameters evaluates to TRUE. The default value is (empty).
Resync_Fails_On_FNF	Determines whether a file-not-found response from the provisioning server constitutes a successful or a failed resync. A failed resync activates the error resync timer. The default value is Yes.
Profile_Rule	This parameter is a profile script that evaluates to the provisioning resync command. The command specifies the protocol (TFTP, HTTP, or HTTPS) and an associated URL. If the command is not specified, TFTP is assumed, and the address of the TFTP server is obtained through DHCP option 66. In the URL, either the IP address or the FQDN of the server can be specified. The file name can have macros, such as $MA, which expands to the device MAC address. The default value is /spa$PSN.cfg.

Parameter Name	Description and Default Value
Profile_Rule_B, Profile_Rule_C, Profile_Rule_D	Defines second, third, and fourth resync commands and associated profile URLs. These profile scripts are executed sequentially after the primary Profile Rule resync operation has completed. If a resync is triggered and Profile Rule is blank, Profile Rule B, C, and D are still evaluated and executed. The default value is (empty).
Log_Resync_Request_Msg	This parameter contains the message that is sent to the syslog server at the start of a resync attempt. The default value is $PN $MAC – Requesting resync $SCHEME:// $SERVIP:$PORT$PATH.
Log_Resync_Success_Msg	The syslog message that is issued upon successful completion of a resync attempt. The default value is $PN $MAC – Successful resync $SCHEME:// $SERVIP:$PORT$PATH -- $ERR.
Log_Resync_Failure_Msg	The syslog message that is issued after a failed resync attempt. The default value is $PN $MAC – Resync failed: $ERR.

Parameter Name	Description and Default Value
Report_Rule	The target URL to which configuration reports are sent. This parameter has the same syntax as the Profile_Rule parameter, and resolves to a TCP/IP command with an associated URL. A configuration report is generated in response to an authenticated SIP NOTIFY message, with Event: report. The report is an XML file containing the name and value of all the device parameters. This parameter may optionally contain an encryption key. For example: [--key $K] tftp://ps.callhome.net/$MA/rep.xml.enc The default value is (empty).

Firmware Upgrade Parameters

The following table defines the function and usage of each parameter in the Firmware Upgrade section of the Provisioning tab.

Parameter Name	Description and Default Value
Upgrade_Enable	Enables firmware upgrade operations independently of resync actions. The default value is Yes.
Upgrade_Error_Retry_Delay	The upgrade retry interval (in seconds) applied in case of upgrade failure. The device has a firmware upgrade error timer that activates after a failed firmware upgrade attempt. The timer is initialized with the value in this parameter. The next firmware upgrade attempt occurs when this timer counts down to zero. The default value is 3600 seconds.
Downgrade_Rev_Limit	Enforces a lower limit on the acceptable version number during a firmware upgrade or downgrade. The device does not complete a firmware upgrade operation unless the firmware version is greater than or equal to this parameter. The default value is (empty).
Upgrade_Rule	This parameter is a firmware upgrade script with the same syntax as Profile_Rule. Defines upgrade conditions and associated firmware URLs. The default value is (empty).
Log_Upgrade_Request_Msg	The syslog message that is issued at the start of a firmware upgrade attempt. The default value is $PN $MAC -- Requesting upgrade $SCHEME://$SERVIP:$PORT$PATH.

Parameter Name	Description and Default Value
Log_Upgrade_Success_Msg	The syslog message that is issued after a firmware upgrade attempt completes successfully. The default value is $PN $MAC -- Successful upgrade $SCHEME://$SERVIP:$PORT$PATH -- $ERR.
Log_Upgrade_Failure_Msg	The syslog message that is issued after a failed firmware upgrade attempt. The default value is $PN $MAC -- Upgrade failed: $ERR.

General Purpose Parameters

The following table defines the function and usage of each parameter in the General Purpose Parameters section of the Provisioning tab.

Parameter Name	Description and Default Value
GPP_SA, GPP_SB, GPP_SC, GPP_SD	Special purpose provisioning parameters, designed to hold encryption keys and passwords. To ensure the integrity of the encryption mechanism, these parameters must be kept secret. Therefore these parameters are not displayed on the device configuration web page, and they are not included in the configuration report sent in response to a SIP NOTIFY command. The default value is (empty).
GPP_A through GPP_P	General purpose provisioning parameters. These parameters can be used as variables in provisioning and upgrade rules. They are referenced by prepending the variable name with a '$' character, such as $GPP_A. The default value is (empty).

Macro Expansion Variables

Certain macro variables are recognized within the following provisioning parameters:

- Profile_Rule
- Profile_Rule_*
- Resync_Trigger_*
- Log_Resync_*
- Upgrade_Rule
- Log_Upgrade_*
- GPP_* (under specific conditions)

Within these parameters, syntax types, such as $NAME or $(NAME), are recognized and expanded.

Macro variable substrings can be specified with the notation $(NAME:p) and $(NAME:p:q), where p and q are non-negative integers (available in revision 2.0.11 and above). The resulting macro expansion is the substring starting at character offset p, with length q (or else till end-of-string if q is not specified). For example, if GPP_A contains ABCDEF, then $(A:2) expands to CDEF, and $(A:2:3) expands to CDE.

An unrecognized name is not translated, and the $NAME or $(NAME) form remains unchanged in the parameter value after expansion.

Parameter Name	Description and Default Value
$	The form $$ expands to a single $ character.
A through P	Replaced by the contents of the general purpose parameters GPP_A through GPP_P.

Parameter Name	Description and Default Value
SA through SD	Replaced by the contents of the special purpose parameters GPP_SA through GPP_SD. These parameters are meant to hold keys or passwords used in provisioning. Note that $SA through $SD are only recognized as arguments to the optional resync URL qualifier **--key**, as in the following example: [--key $SA] http://ps.callme.com/profiles/abcdefg.cfg These variables are not expanded outside of this limited context.
MA	MAC address using lower case hex digits, for example, 000e08aabbcc.
MAU	MAC address using upper case hex digits, for example 000E08AABBCC.
MAC	MAC address using lower case hex digits, and colons to separate hex digit pairs, for example 00:0e:08:aa:bb:cc.
PN	Product Name, for example SPA962.
PSN	Product Series Number, for example 962.
SN	Serial Number string, for example 88012BA01234.
CCERT	SSL Client Certificate status: Installed or Not Installed.
IP	IP address of the IP Telephony Device within its local subnet, for example 192.168.1.100.
EXTIP	External IP of the IP Telephony Device, as seen on the Internet, for example 66.43.16.52.
SWVER	Software version string, for example 2.0.6(b).
HWVER	Hardware version string, for example 1.88.1.

Parameter Name	Description and Default Value
PRVST	Provisioning State, a numeric string: -1 = explicit resync request, 0 = power-up resync, 1 = periodic resync, 2 = resync failed, retry attempt
UPGST	Upgrade State, a numeric string: 1 = first upgrade attempt, 2 = upgrade failed, retry attempt
UPGERR	Result message (ERR) of previous upgrade attempt, for example http_get failed.
PRVTMR	Seconds since last resync attempt.
UPGTMR	Seconds since last upgrade attempt.
REGTMR1	Seconds since Line 1 lost registration with SIP server.
REGTMR2	Seconds since Line 2 lost registration with SIP server.
UPGCOND	Legacy macro name, always expands to true in firmware rev 2.0.6 and above.
SCHEME	File access scheme, one of TFTP, HTTP, or HTTPS, as obtained after parsing resync or upgrade URL.
METH	Deprecated alias for SCHEME, do not use.
SERV	Request target server host name, as obtained after parsing resync or upgrade URL.
SERVIP	Request target server IP address, as obtained after parsing resync or upgrade URL, possibly following DNS lookup.
PORT	Request target UDP/TCP port, as obtained after parsing resync or upgrade URL.
PATH	Request target file path, as obtained after parsing resync or upgrade URL.

Parameter Name	Description and Default Value
ERR	Result message of resync or upgrade attempt. Only useful in generating result syslog messages. The value is preserved in the UPGERR variable in the case of upgrade attempts.
UID1	The contents of the Line 1 User_ID configuration parameter (Firmware 2.0.11 and above).
UID2	The contents of the Line 2 User_ID configuration parameter (Firmware 2.0.11 and above).
ISCUST	Value=1 if unit is customized, 0 otherwise; customization status viewable on WebUI Info page.

Internal Error Codes

The IP Telephony Device defines a number of internal error codes (X00–X99) to facilitate configuration in providing finer control over the behavior of the unit under certain error conditions.

Parameter Name	Description and Default Value
X00	Transport layer (or ICMP) error when sending a SIP request.
X20	SIP request times out while waiting for a response.
X40	General SIP protocol error (for example, unacceptable codec in SDP in 200 and ACK messages, or times out while waiting for ACK).
X60	Dialed number invalid according to given dial plan.

A

Example Configuration Profile

What follows is a *sample* profile. An up-to-date profile template can be obtained from the SPC tool, with the command line invocation spc --sample-profile sample.txt.

```
# ***
# *** Linksys SPA Series Configuration Parameters
# ***


# *** System Configuration

Restricted_Access_Domains          "" ;
Enable_Web_Server                  "Yes" ;
Web_Server_Port                    "80" ;
Enable_Web_Admin_Access            "Yes" ;
Admin_Passwd                       "" ;
User_Password                    ! "" ;

# *** Internet Connection Type

DHCP                             ! "Yes" ;
Static_IP                        ! "" ;
NetMask                          ! "" ;
Gateway                          ! "" ;

# *** Optional Network Configuration

HostName                         ! "" ;
Domain                           ! "" ;
Primary_DNS                      ! "" ;
Secondary_DNS                    ! "" ;
DNS_Server_Order                   "Manual" ;  # options: Manual/Manual,DHCP/
DHCP,Manual
DNS_Query_Mode                    "Parallel" ;  # options: Parallel/Sequential
Syslog_Server                      "" ;
Debug_Server                       "" ;
Debug_Level                        "0" ;  # options: 0/1/2/3
Primary_NTP_Server                 "" ;
Secondary_NTP_Server               "" ;

# *** Configuration Profile

Provision_Enable                   "Yes" ;
Resync_On_Reset                    "Yes" ;
```

```
Resync_Random_Delay                "2" ;
Resync_Periodic                    "3600" ;
Resync_Error_Retry_Delay           "3600" ;
Forced_Resync_Delay                "14400" ;
Resync_From_SIP                    "Yes" ;
Resync_After_Upgrade_Attempt       "Yes" ;
Resync_Trigger_1                   "" ;
Resync_Trigger_2                   "" ;
Profile_Rule                       "/spa$PSN.cfg" ;
Profile_Rule_B                     "" ;
Profile_Rule_C                     "" ;
Profile_Rule_D                     "" ;
Log_Resync_Request_Msg             "$PN $MAC -- Requesting resync $SCHEME://
$SERVIP:$PORT$PATH" ;
Log_Resync_Success_Msg             "$PN $MAC -- Successful resync $SCHEME://
$SERVIP:$PORT$PATH" ;
Log_Resync_Failure_Msg             "$PN $MAC -- Resync failed: $ERR" ;

# *** Firmware Upgrade

Upgrade_Enable                     "Yes" ;
Upgrade_Error_Retry_Delay          "3600" ;
Downgrade_Rev_Limit                "" ;
Upgrade_Rule                       "" ;
Log_Upgrade_Request_Msg            "$PN $MAC -- Requesting upgrade $SCHEME://
$SERVIP:$PORT$PATH" ;
Log_Upgrade_Success_Msg            "$PN $MAC -- Successful upgrade $SCHEME://
$SERVIP:$PORT$PATH -- $ERR" ;
Log_Upgrade_Failure_Msg            "$PN $MAC -- Upgrade failed: $ERR" ;

# *** General Purpose Parameters

GPP_A                              "" ;
GPP_B                              "" ;
GPP_C                              "" ;
GPP_D                              "" ;
GPP_E                              "" ;
GPP_F                              "" ;
GPP_G                              "" ;
GPP_H                              "" ;
GPP_I                              "" ;
GPP_J                              "" ;
GPP_K                              "" ;
GPP_L                              "" ;
GPP_M                              "" ;
GPP_N                              "" ;
GPP_O                              "" ;
GPP_P                              "" ;
GPP_SA                             "" ;
GPP_SB                             "" ;
GPP_SC                             "" ;
GPP_SD                             "" ;

# *** SIP Parameters
```

```
Max_Forward                     "70" ;
Max_Redirection                 "5" ;
Max_Auth                        "2" ;
SIP_User_Agent_Name             "$VERSION" ;
SIP_Server_Name                 "$VERSION" ;
SIP_Accept_Language             "" ;
DTMF_Relay_MIME_Type            "application/dtmf-relay" ;
Hook_Flash_MIME_Type            "application/hook-flash" ;
Remove_Last_Reg                 "No" ;
Use_Compact_Header              "No" ;

# *** SIP Timer Values (sec)

SIP_T1                          ".5" ;
SIP_T2                          "4" ;
SIP_T4                          "5" ;
SIP_Timer_B                     "32" ;
SIP_Timer_F                     "32" ;
SIP_Timer_H                     "32" ;
SIP_Timer_D                     "32" ;
SIP_Timer_J                     "32" ;
INVITE_Expires                  "240" ;
ReINVITE_Expires                "30" ;
Reg_Min_Expires                 "1" ;
Reg_Max_Expires                 "7200" ;
Reg_Retry_Intvl                 "30" ;
Reg_Retry_Long_Intvl            "1200" ;

# *** Response Status Code Handling

SIT1_RSC                        "" ;
SIT2_RSC                        "" ;
SIT3_RSC                        "" ;
SIT4_RSC                        "" ;
Try_Backup_RSC                  "" ;
Retry_Reg_RSC                   "" ;

# *** RTP Parameters

RTP_Port_Min                    "16384" ;
RTP_Port_Max                    "16482" ;
RTP_Packet_Size                 "0.030" ;
Max_RTP_ICMP_Err                "0" ;
RTCP_Tx_Interval                "0" ;

# *** SDP Payload Types

NSE_Dynamic_Payload             "100" ;
AVT_Dynamic_Payload             "101" ;
G726r16_Dynamic_Payload         "98" ;
G726r24_Dynamic_Payload         "97" ;
G726r40_Dynamic_Payload         "96" ;
G729b_Dynamic_Payload           "99" ;
NSE_Codec_Name                  "NSE" ;
AVT_Codec_Name                  "telephone-event" ;
```

```
G711u_Codec_Name                "PCMU" ;
G711a_Codec_Name                "PCMA" ;
G726r16_Codec_Name              "G726-16" ;
G726r24_Codec_Name              "G726-24" ;
G726r32_Codec_Name              "G726-32" ;
G726r40_Codec_Name              "G726-40" ;
G729a_Codec_Name                "G729a" ;
G729b_Codec_Name                "G729ab" ;
G723_Codec_Name                 "G723" ;

# *** NAT Support Parameters

Handle_VIA_received             "No" ;
Handle_VIA_rport                "No" ;
Insert_VIA_received             "No" ;
Insert_VIA_rport                "No" ;
Substitute_VIA_Addr             "No" ;
Send_Resp_To_Src_Port           "No" ;
STUN_Enable                     "No" ;
STUN_Test_Enable                "No" ;
STUN_Server                     "" ;
EXT_IP                          "" ;
EXT_RTP_Port_Min                "" ;
NAT_Keep_Alive_Intvl            "15" ;

# ***

Line_Enable[1]                  "Yes" ;
SAS_Enable[1]                   "No" ;
MOH_Server[1]                   "" ;
SAS_DLG_Refresh_Intvl[1]        "30" ;
NAT_Mapping_Enable[1]           "No" ;
SAS_Inbound_RTP_Sink[1]         "" ;
SIP_Port[1]                     "5060" ;
NAT_Keep_Alive_Enable[1]        "No" ;
EXT_SIP_Port[1]                 "" ;
NAT_Keep_Alive_Msg[1]           "$NOTIFY" ;
SIP_TOS/DiffServ_Value[1]       "0x68" ;
NAT_Keep_Alive_Dest[1]          "$PROXY" ;
RTP_TOS/DiffServ_Value[1]       "0xb8" ;
SIP_Debug_Option[1]             "none" ;  # options: none/1-line/1-line
excl. OPT/1-line excl. NTFY/1-line excl. REG/1-line excl. OPT|NTFY|REG/full/
full excl. OPT/full excl. NTFY/full excl. REG/full excl. OPT|NTFY|REG
Network_Jitter_Level[1]         "high" ;  # options: low/medium/high/very
high
SIP_100REL_Enable[1]            "No" ;
Blind_Attn-Xfer_Enable[1]       "No" ;
SIP_Proxy-Require[1]            "" ;
Auth_Resync-Reboot[1]           "Yes" ;
SIP_Remote-Party-ID[1]          "No" ;

# *** Proxy and Registration

Proxy[1]                        "" ;
Use_Outbound_Proxy[1]           "No" ;
```

```
Outbound_Proxy[1]                 "" ;
Use_OB_Proxy_In_Dialog[1]         "Yes" ;
Register[1]                       "Yes" ;
Make_Call_Without_Reg[1]          "No" ;
Register_Expires[1]               "3600" ;
Ans_Call_Without_Reg[1]           "No" ;
Use_DNS_SRV[1]                    "No" ;
DNS_SRV_Auto_Prefix[1]            "No" ;
Proxy_Fallback_Intvl[1]           "3600" ;

# *** Subscriber Information

Display_Name[1]                   "" ;
User_ID[1]                        "" ;
Password[1]                       "" ;
Use_Auth_ID[1]                    "No" ;
Auth_ID[1]                        "" ;
Mini_Certificate[1]               "" ;
SRTP_Private_Key[1]               "" ;

# *** Supplementary Service Subscription

Call_Waiting_Serv[1]              "Yes" ;
Block_CID_Serv[1]                 "Yes" ;
Block_ANC_Serv[1]                 "Yes" ;
Dist_Ring_Serv[1]                 "Yes" ;
Cfwd_All_Serv[1]                  "Yes" ;
Cfwd_Busy_Serv[1]                 "Yes" ;
Cfwd_No_Ans_Serv[1]               "Yes" ;
Cfwd_Sel_Serv[1]                  "Yes" ;
Cfwd_Last_Serv[1]                 "Yes" ;
Block_Last_Serv[1]                "Yes" ;
Accept_Last_Serv[1]               "Yes" ;
DND_Serv[1]                       "Yes" ;
CID_Serv[1]                       "Yes" ;
CWCID_Serv[1]                     "Yes" ;
Call_Return_Serv[1]               "Yes" ;
Call_Back_Serv[1]                 "Yes" ;
Three_Way_Call_Serv[1]            "Yes" ;
Three_Way_Conf_Serv[1]            "Yes" ;
Attn_Transfer_Serv[1]             "Yes" ;
Unattn_Transfer_Serv[1]           "Yes" ;
MWI_Serv[1]                       "Yes" ;
VMWI_Serv[1]                      "Yes" ;
Speed_Dial_Serv[1]                "Yes" ;
Secure_Call_Serv[1]               "Yes" ;
Referral_Serv[1]                  "Yes" ;
Feature_Dial_Serv[1]              "Yes" ;

# *** Audio Configuration

Preferred_Codec[1]                "G711u" ;  # options: G711u/G711a/G726-16/
G726-24/G726-32/G726-40/G729a/G723
Silence_Supp_Enable[1]            "No" ;
Use_Pref_Codec_Only[1]            "No" ;
```

```
Echo_Canc_Enable[1]                "Yes" ;
G729a_Enable[1]                    "Yes" ;
Echo_Canc_Adapt_Enable[1]          "Yes" ;
G723_Enable[1]                     "Yes" ;
Echo_Supp_Enable[1]                "Yes" ;
G726-16_Enable[1]                  "Yes" ;
FAX_CED_Detect_Enable[1]           "Yes" ;
G726-24_Enable[1]                  "Yes" ;
FAX_CNG_Detect_Enable[1]           "Yes" ;
G726-32_Enable[1]                  "Yes" ;
FAX_Passthru_Codec[1]              "G711u" ;  # options: G711u/G711a
G726-40_Enable[1]                  "Yes" ;
FAX_Codec_Symmetric[1]             "Yes" ;
DTMF_Tx_Method[1]                  "Auto" ;  # options: InBand/AVT/INFO/Auto
FAX_Passthru_Method[1]             "NSE" ;  # options: None/NSE/ReINVITE
Hook_Flash_Tx_Method[1]            "None" ;  # options: None/AVT/INFO
FAX_Process_NSE[1]                 "Yes" ;
Release_Unused_Codec[1]            "Yes" ;

# *** Dial Plan

Dial_Plan[1]                       "(*xx|[3469]11|0|00|[2-9]xxxxxx|1xxx[2-
9]xxxxxxS0|xxxxxxxxxxx.)" ;
Enable_IP_Dialing[1]               "No" ;

# *** FXS Port Polarity Configuration

Idle_Polarity[1]                   "Forward" ;  # options: Forward/Reverse
Caller_Conn_Polarity[1]            "Forward" ;  # options: Forward/Reverse
Callee_Conn_Polarity[1]            "Forward" ;  # options: Forward/Reverse

# *** Call Forward Settings

Cfwd_All_Dest[1]                 ! "" ;
Cfwd_Busy_Dest[1]                ! "" ;
Cfwd_No_Ans_Dest[1]              ! "" ;
Cfwd_No_Ans_Delay[1]             ! "20" ;

# *** Selective Call Forward Settings

Cfwd_Sel1_Caller[1]              ! "" ;
Cfwd_Sel1_Dest[1]                ! "" ;
Cfwd_Sel2_Caller[1]              ! "" ;
Cfwd_Sel2_Dest[1]                ! "" ;
Cfwd_Sel3_Caller[1]              ! "" ;
Cfwd_Sel3_Dest[1]                ! "" ;
Cfwd_Sel4_Caller[1]              ! "" ;
Cfwd_Sel4_Dest[1]                ! "" ;
Cfwd_Sel5_Caller[1]              ! "" ;
Cfwd_Sel5_Dest[1]                ! "" ;
Cfwd_Sel6_Caller[1]              ! "" ;
Cfwd_Sel6_Dest[1]                ! "" ;
Cfwd_Sel7_Caller[1]              ! "" ;
Cfwd_Sel7_Dest[1]                ! "" ;
Cfwd_Sel8_Caller[1]              ! "" ;
```

```
Cfwd_Sel8_Dest[1]               ! "" ;
Cfwd_Last_Caller[1]             ! "" ;
Cfwd_Last_Dest[1]               ! "" ;
Block_Last_Caller[1]            ! "" ;
Accept_Last_Caller[1]           ! "" ;

# *** Speed Dial Settings

Speed_Dial_2[1]                 ! "" ;
Speed_Dial_3[1]                 ! "" ;
Speed_Dial_4[1]                 ! "" ;
Speed_Dial_5[1]                 ! "" ;
Speed_Dial_6[1]                 ! "" ;
Speed_Dial_7[1]                 ! "" ;
Speed_Dial_8[1]                 ! "" ;
Speed_Dial_9[1]                 ! "" ;

# *** Supplementary Service Settings

CW_Setting[1]                   ! "Yes" ;
Block_CID_Setting[1]            ! "No" ;
Block_ANC_Setting[1]            ! "No" ;
DND_Setting[1]                  ! "No" ;
CID_Setting[1]                  ! "Yes" ;
CWCID_Setting[1]                ! "Yes" ;
Dist_Ring_Setting[1]            ! "Yes" ;
Secure_Call_Setting[1]            "No" ;

# *** Distinctive Ring Settings

Ring1_Caller[1]                 ! "" ;
Ring2_Caller[1]                 ! "" ;
Ring3_Caller[1]                 ! "" ;
Ring4_Caller[1]                 ! "" ;
Ring5_Caller[1]                 ! "" ;
Ring6_Caller[1]                 ! "" ;
Ring7_Caller[1]                 ! "" ;
Ring8_Caller[1]                 ! "" ;

# *** Ring Settings

Default_Ring[1]                 ! "1" ;  # options: 1/2/3/4/5/6/7/8
Default_CWT[1]                  ! "1" ;  # options: 1/2/3/4/5/6/7/8
Hold_Reminder_Ring[1]           ! "8" ;  # options: 1/2/3/4/5/6/7/8/none
Call_Back_Ring[1]               ! "7" ;  # options: 1/2/3/4/5/6/7/8
Cfwd_Ring_Splash_Len[1]         ! "0" ;
Cblk_Ring_Splash_Len[1]         ! "0" ;
VMWI_Ring_Splash_Len[1]         ! ".5" ;
VMWI_Ring_Policy[1]               "New VM Available" ;  # options: New VM
Available/New VM Becomes Available/New VM Arrives
Ring_On_No_New_VM[1]              "No" ;

# ***

Line_Enable[2]                    "Yes" ;
```

```
SAS_Enable[2]                    "No" ;
MOH_Server[2]                    "" ;
SAS_DLG_Refresh_Intvl[2]         "30" ;
NAT_Mapping_Enable[2]            "No" ;
SAS_Inbound_RTP_Sink[2]          "" ;
SIP_Port[2]                      "5061" ;
NAT_Keep_Alive_Enable[2]         "No" ;
EXT_SIP_Port[2]                  "" ;
NAT_Keep_Alive_Msg[2]            "$NOTIFY" ;
SIP_TOS/DiffServ_Value[2]        "0x68" ;
NAT_Keep_Alive_Dest[2]           "$PROXY" ;
RTP_TOS/DiffServ_Value[2]        "0xb8" ;
SIP_Debug_Option[2]              "none" ;  # options: none/1-line/1-line
excl. OPT/1-line excl. NTFY/1-line excl. REG/1-line excl. OPT|NTFY|REG/full/
full excl. OPT/full excl. NTFY/full excl. REG/full excl. OPT|NTFY|REG
Network_Jitter_Level[2]          "high" ;  # options: low/medium/high/very
high
SIP_100REL_Enable[2]             "No" ;
Blind_Attn-Xfer_Enable[2]        "No" ;
SIP_Proxy-Require[2]             "" ;
Auth_Resync-Reboot[2]            "Yes" ;
SIP_Remote-Party-ID[2]           "No" ;

# *** Proxy and Registration

Proxy[2]                         "" ;
Use_Outbound_Proxy[2]            "No" ;
Outbound_Proxy[2]                "" ;
Use_OB_Proxy_In_Dialog[2]        "Yes" ;
Register[2]                      "Yes" ;
Make_Call_Without_Reg[2]         "No" ;
Register_Expires[2]              "3600" ;
Ans_Call_Without_Reg[2]          "No" ;
Use_DNS_SRV[2]                   "No" ;
DNS_SRV_Auto_Prefix[2]           "No" ;
Proxy_Fallback_Intvl[2]          "3600" ;

# *** Subscriber Information

Display_Name[2]                  "" ;
User_ID[2]                       "" ;
Password[2]                      "" ;
Use_Auth_ID[2]                   "No" ;
Auth_ID[2]                       "" ;
Mini_Certificate[2]              "" ;
SRTP_Private_Key[2]              "" ;

# *** Supplementary Service Subscription

Call_Waiting_Serv[2]             "Yes" ;
Block_CID_Serv[2]                "Yes" ;
Block_ANC_Serv[2]                "Yes" ;
Dist_Ring_Serv[2]                "Yes" ;
Cfwd_All_Serv[2]                 "Yes" ;
Cfwd_Busy_Serv[2]                "Yes" ;
```

```
Cfwd_No_Ans_Serv[2]                "Yes" ;
Cfwd_Sel_Serv[2]                   "Yes" ;
Cfwd_Last_Serv[2]                  "Yes" ;
Block_Last_Serv[2]                 "Yes" ;
Accept_Last_Serv[2]                "Yes" ;
DND_Serv[2]                        "Yes" ;
CID_Serv[2]                        "Yes" ;
CWCID_Serv[2]                      "Yes" ;
Call_Return_Serv[2]                "Yes" ;
Call_Back_Serv[2]                  "Yes" ;
Three_Way_Call_Serv[2]             "Yes" ;
Three_Way_Conf_Serv[2]             "Yes" ;
Attn_Transfer_Serv[2]              "Yes" ;
Unattn_Transfer_Serv[2]            "Yes" ;
MWI_Serv[2]                        "Yes" ;
VMWI_Serv[2]                       "Yes" ;
Speed_Dial_Serv[2]                 "Yes" ;
Secure_Call_Serv[2]                "Yes" ;
Referral_Serv[2]                   "Yes" ;
Feature_Dial_Serv[2]               "Yes" ;

# *** Audio Configuration

Preferred_Codec[2]                 "G711u" ;  # options: G711u/G711a/G726-16/
G726-24/G726-32/G726-40/G729a/G723
Silence_Supp_Enable[2]             "No" ;
Use_Pref_Codec_Only[2]             "No" ;
Echo_Canc_Enable[2]                "Yes" ;
G729a_Enable[2]                    "Yes" ;
Echo_Canc_Adapt_Enable[2]          "Yes" ;
G723_Enable[2]                     "Yes" ;
Echo_Supp_Enable[2]                "Yes" ;
G726-16_Enable[2]                  "Yes" ;
FAX_CED_Detect_Enable[2]           "Yes" ;
G726-24_Enable[2]                  "Yes" ;
FAX_CNG_Detect_Enable[2]           "Yes" ;
G726-32_Enable[2]                  "Yes" ;
FAX_Passthru_Codec[2]              "G711u" ;  # options: G711u/G711a
G726-40_Enable[2]                  "Yes" ;
FAX_Codec_Symmetric[2]             "Yes" ;
DTMF_Tx_Method[2]                  "Auto" ;  # options: InBand/AVT/INFO/Auto
FAX_Passthru_Method[2]             "NSE" ;  # options: None/NSE/ReINVITE
Hook_Flash_Tx_Method[2]            "None" ;  # options: None/AVT/INFO
FAX_Process_NSE[2]                 "Yes" ;
Release_Unused_Codec[2]            "Yes" ;

# *** Dial Plan

Dial_Plan[2]                       "(*xx|[3469]11|0|00|[2-9]xxxxxx|1xxx[2-
9]xxxxxxS0|xxxxxxxxxxx.)" ;
Enable_IP_Dialing[2]               "No" ;

# *** FXS Port Polarity Configuration

Idle_Polarity[2]                   "Forward" ;  # options: Forward/Reverse
```

```
Caller_Conn_Polarity[2]           "Forward" ;  # options: Forward/Reverse
Callee_Conn_Polarity[2]           "Forward" ;  # options: Forward/Reverse

# *** Call Forward Settings

Cfwd_All_Dest[2]                ! "" ;
Cfwd_Busy_Dest[2]               ! "" ;
Cfwd_No_Ans_Dest[2]             ! "" ;
Cfwd_No_Ans_Delay[2]            ! "20" ;

# *** Selective Call Forward Settings

Cfwd_Sel1_Caller[2]             ! "" ;
Cfwd_Sel1_Dest[2]               ! "" ;
Cfwd_Sel2_Caller[2]             ! "" ;
Cfwd_Sel2_Dest[2]               ! "" ;
Cfwd_Sel3_Caller[2]             ! "" ;
Cfwd_Sel3_Dest[2]               ! "" ;
Cfwd_Sel4_Caller[2]             ! "" ;
Cfwd_Sel4_Dest[2]               ! "" ;
Cfwd_Sel5_Caller[2]             ! "" ;
Cfwd_Sel5_Dest[2]               ! "" ;
Cfwd_Sel6_Caller[2]             ! "" ;
Cfwd_Sel6_Dest[2]               ! "" ;
Cfwd_Sel7_Caller[2]             ! "" ;
Cfwd_Sel7_Dest[2]               ! "" ;
Cfwd_Sel8_Caller[2]             ! "" ;
Cfwd_Last_Caller[2]             ! "" ;Cfwd_Sel8_Dest[2]              ! "" ;
Cfwd_Last_Dest[2]               ! "" ;
Block_Last_Caller[2]            ! "" ;
Accept_Last_Caller[2]           ! "" ;

# *** Speed Dial Settings

Speed_Dial_2[2]                 ! "" ;
Speed_Dial_3[2]                 ! "" ;
Speed_Dial_4[2]                 ! "" ;
Speed_Dial_5[2]                 ! "" ;
Speed_Dial_6[2]                 ! "" ;
Speed_Dial_7[2]                 ! "" ;
Speed_Dial_8[2]                 ! "" ;
Speed_Dial_9[2]                 ! "" ;

# *** Supplementary Service Settings

CW_Setting[2]                   ! "Yes" ;
Block_CID_Setting[2]            ! "No" ;
Block_ANC_Setting[2]            ! "No" ;
DND_Setting[2]                  ! "No" ;
CID_Setting[2]                  ! "Yes" ;
CWCID_Setting[2]                ! "Yes" ;
Dist_Ring_Setting[2]            ! "Yes" ;
Secure_Call_Setting[2]            "No" ;

# *** Distinctive Ring Settings
```

```
Ring1_Caller[2]                ! "" ;
Ring2_Caller[2]                ! "" ;
Ring3_Caller[2]                ! "" ;
Ring4_Caller[2]                ! "" ;
Ring5_Caller[2]                ! "" ;
Ring6_Caller[2]                ! "" ;
Ring7_Caller[2]                ! "" ;
Ring8_Caller[2]                ! "" ;

# *** Ring Settings

Default_Ring[2]                ! "1" ;  # options: 1/2/3/4/5/6/7/8
Default_CWT[2]                 ! "1" ;  # options: 1/2/3/4/5/6/7/8
Hold_Reminder_Ring[2]          ! "8" ;  # options: 1/2/3/4/5/6/7/8/none
Call_Back_Ring[2]              ! "7" ;  # options: 1/2/3/4/5/6/7/8
Cfwd_Ring_Splash_Len[2]        ! "0" ;
Cblk_Ring_Splash_Len[2]        ! "0" ;
VMWI_Ring_Splash_Len[2]        ! ".5" ;
VMWI_Ring_Policy[2]             "New VM Available" ;  # options: New VM
Available/New VM Becomes Available/New VM Arrives
Ring_On_No_New_VM[2]            "No" ;

# *** Call Progress Tones

Dial_Tone                       "350@-19,440@-19;10(*/0/1+2)" ;
Second_Dial_Tone                "420@-19,520@-19;10(*/0/1+2)" ;
Outside_Dial_Tone               "420@-16;10(*/0/1)" ;
Prompt_Tone                     "520@-19,620@-19;10(*/0/1+2)" ;
Busy_Tone                       "480@-19,620@-19;10(.5/.5/1+2)" ;
Reorder_Tone                    "480@-19,620@-19;10(.25/.25/1+2)" ;
Off_Hook_Warning_Tone           "480@-10,620@0;10(.125/.125/1+2)" ;
Ring_Back_Tone                  "440@-19,480@-19;*(2/4/1+2)" ;
Confirm_Tone                    "600@-16;1(.25/.25/1)" ;
SIT1_Tone                      "985@-16,1428@-16,1777@-16;20(.380/0/1,.380/
0/2,.380/0/3,0/4/0)" ;
SIT2_Tone                      "914@-16,1371@-16,1777@-16;20(.274/0/1,.274/
0/2,.380/0/3,0/4/0)" ;
SIT3_Tone                      "914@-16,1371@-16,1777@-16;20(.380/0/1,.380/
0/2,.380/0/3,0/4/0)" ;
SIT4_Tone                      "985@-16,1371@-16,1777@-16;20(.380/0/1,.274/
0/2,.380/0/3,0/4/0)" ;
MWI_Dial_Tone                  "350@-19,440@-19;2(.1/.1/1+2);10(*/0/1+2)" ;
Cfwd_Dial_Tone                 "350@-19,440@-19;2(.2/.2/1+2);10(*/0/1+2)" ;
Holding_Tone                    "600@-19;*(.1/.1/1,.1/.1/1,.1/9.5/1)" ;
Conference_Tone                 "350@-19;20(.1/.1/1,.1/9.7/1)" ;
Secure_Call_Indication_Tone     "397@-19,507@-19;15(0/2/0,.2/.1/1,.1/2.1/
2)" ;

# *** Distinctive Ring Patterns

Ring1_Cadence                   "60(2/4)" ;
Ring2_Cadence                   "60(.3/.2,1/.2,.3/4)" ;
Ring3_Cadence                   "60(.8/.4,.8/4)" ;
Ring4_Cadence                   "60(.4/.2,.3/.2,.8/4)" ;
```

```
Ring5_Cadence                    "60(.2/.2,.2/.2,.2/.2,1/4)" ;
Ring6_Cadence                    "60(.2/.4,.2/.4,.2/4)" ;
Ring7_Cadence                    "60(.4/.2,.4/.2,.4/4)" ;
Ring8_Cadence                    "60(0.25/9.75)" ;

# *** Distinctive Call Waiting Tone Patterns

CWT1_Cadence                     "30(.3/9.7)" ;
CWT2_Cadence                     "30(.1/.1, .1/9.7)" ;
CWT3_Cadence                     "30(.1/.1, .3/.1, .1/9.3)" ;
CWT4_Cadence                     "30(.1/.1,.1/.1,.1/9.5)" ;
CWT5_Cadence                     "30(.3/.1,.1/.1,.3/9.1)" ;
CWT6_Cadence                     "30(.1/.1,.3/.2,.3/9.1)" ;
CWT7_Cadence                     "30(.3/.1,.3/.1,.1/9.1)" ;
CWT8_Cadence                     "2.3(.3/2)" ;

# *** Distinctive Ring/CWT Pattern Names

Ring1_Name                       "Bellcore-r1" ;
Ring2_Name                       "Bellcore-r2" ;
Ring3_Name                       "Bellcore-r3" ;
Ring4_Name                       "Bellcore-r4" ;
Ring5_Name                       "Bellcore-r5" ;
Ring6_Name                       "Bellcore-r6" ;
Ring7_Name                       "Bellcore-r7" ;
Ring8_Name                       "Bellcore-r8" ;

# *** Ring and Call Waiting Tone Spec

Ring_Waveform                    "Sinusoid" ;  # options: Sinusoid/Trapezoid
Ring_Frequency                   "25" ;
Ring_Voltage                     "70" ;
CWT_Frequency                    "440@-10" ;

# *** Control Timer Values (sec)

Hook_Flash_Timer_Min             ".1" ;
Hook_Flash_Timer_Max             ".9" ;
Callee_On_Hook_Delay             "0" ;
Reorder_Delay                    "5" ;
Call_Back_Expires                "1800" ;
Call_Back_Retry_Intvl            "30" ;
Call_Back_Delay                  ".5" ;
VMWI_Refresh_Intvl               "30" ;
Interdigit_Long_Timer            "10" ;
Interdigit_Short_Timer           "3" ;
CPC_Delay                        "2" ;
CPC_Duration                     "0" ;

# *** Vertical Service Activation Codes

Call_Return_Code                 "*69" ;
Blind_Transfer_Code              "*98" ;
Call_Back_Act_Code               "*66" ;
Call_Back_Deact_Code             "*86" ;
```

```
Cfwd_All_Act_Code                "*72" ;
Cfwd_All_Deact_Code              "*73" ;
Cfwd_Busy_Act_Code               "*90" ;
Cfwd_Busy_Deact_Code             "*91" ;
Cfwd_No_Ans_Act_Code             "*92" ;
Cfwd_No_Ans_Deact_Code           "*93" ;
Cfwd_Last_Act_Code               "*63" ;
Cfwd_Last_Deact_Code             "*83" ;
Block_Last_Act_Code              "*60" ;
Block_Last_Deact_Code            "*80" ;
Accept_Last_Act_Code             "*64" ;
Accept_Last_Deact_Code           "*84" ;
CW_Act_Code                      "*56" ;
CW_Deact_Code                    "*57" ;
CW_Per_Call_Act_Code             "*71" ;
CW_Per_Call_Deact_Code           "*70" ;
Block_CID_Act_Code               "*67" ;
Block_CID_Deact_Code             "*68" ;
Block_CID_Per_Call_Act_Code      "*81" ;
Block_CID_Per_Call_Deact_Code    "*82" ;
Block_ANC_Act_Code               "*77" ;
Block_ANC_Deact_Code             "*87" ;
DND_Act_Code                     "*78" ;
DND_Deact_Code                   "*79" ;
CID_Act_Code                     "*65" ;
CID_Deact_Code                   "*85" ;
CWCID_Act_Code                   "*25" ;
CWCID_Deact_Code                 "*45" ;
Dist_Ring_Act_Code               "*26" ;
Dist_Ring_Deact_Code             "*46" ;
Speed_Dial_Act_Code              "*74" ;
Secure_All_Call_Act_Code         "*16" ;
Secure_No_Call_Act_Code          "*17" ;
Secure_One_Call_Act_Code         "*18" ;
Secure_One_Call_Deact_Code       "*19" ;
Referral_Services_Codes          "" ;
Feature_Dial_Services_Codes      "" ;

# *** Outbound Call Codec Selection Codes

Prefer_G711u_Code                "*017110" ;
Force_G711u_Code                 "*027110" ;
Prefer_G711a_Code                "*017111" ;
Force_G711a_Code                 "*027111" ;
Prefer_G723_Code                 "*01723" ;
Force_G723_Code                  "*02723" ;
Prefer_G726r16_Code              "*0172616" ;
Force_G726r16_Code               "*0272616" ;
Prefer_G726r24_Code              "*0172624" ;
Force_G726r24_Code               "*0272624" ;
Prefer_G726r32_Code              "*0172632" ;
Force_G726r32_Code               "*0272632" ;
Prefer_G726r40_Code              "*0172640" ;
Force_G726r40_Code               "*0272640" ;
Prefer_G729a_Code                "*01729" ;
```

```
Force_G729a_Code                "*02729" ;

# *** Miscellaneous

Set_Local_Date_(mm/dd)          "" ;
Set_Local_Time_(HH/mm)          "" ;
Time_Zone                       "GMT-07:00" ;  # options: GMT-12:00/GMT-
11:00/GMT-10:00/GMT-09:00/GMT-08:00/GMT-07:00/GMT-06:00/GMT-05:00/GMT-04:00/
GMT-03:30/GMT-03:00/GMT-02:00/GMT-01:00/GMT/GMT+01:00/GMT+02:00/GMT+03:00/
GMT+03:30/GMT+04:00/GMT+05:00/GMT+05:30/GMT+05:45/GMT+06:00/GMT+06:30/
GMT+07:00/GMT+08:00/GMT+09:00/GMT+09:30/GMT+10:00/GMT+11:00/GMT+12:00/
GMT+13:00
FXS_Port_Impedance              "600" ;  # options: 600/900/600+2.16uF/
900+2.16uF/270+750||150nF/220+820||120nF/220+820||115nF/370+620||310nF
FXS_Port_Input_Gain             "-3" ;
FXS_Port_Output_Gain            "-3" ;
DTMF_Playback_Level             "-16" ;
DTMF_Playback_Length            ".1" ;
Detect_ABCD                     "Yes" ;
Playback_ABCD                   "Yes" ;
Caller_ID_Method                "Bellcore(N.Amer,China)" ;  # options:
Bellcore(N.Amer,China)/DTMF(Finland,Sweden)/DTMF(Denmark)/ETSI DTMF/ETSI
DTMF With PR/ETSI DTMF After Ring/ETSI FSK/ETSI FSK With PR(UK)
FXS_Port_Power_Limit            "3" ;  # options: 1/2/3/4/5/6/7/8
Protect_IVR_FactoryReset        "No" ;
```

B

Acronyms

A/D	Analog To Digital Converter
ANC	Anonymous Call
B2BUA	Back to Back User Agent
Bool	Boolean Values. Specified as yes and no, or 1 and 0 in the profile
CA	Certificate Authority
CAS	CPE Alert Signal
CDR	Call Detail Record
CID	Caller ID
CIDCW	Call Waiting Caller ID
CNG	Comfort Noise Generation
CPC	Calling Party Control
CPE	Customer Premises Equipment
CWCID	Call Waiting Caller ID
CWT	Call Waiting Tone
D/A	Digital to Analog Converter
dB	decibel
dBm	dB with respect to 1 milliwatt
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System

DRAM	Dynamic Random Access Memory
DSL	Digital Subscriber Loop
DSP	Digital Signal Processor
DTAS	Data Terminal Alert Signal (same as CAS)
DTMF	Dual Tone Multiple Frequency
FQDN	Fully Qualified Domain Name
FSK	Frequency Shift Keying
FXS	Foreign eXchange Station
GW	Gateway
ITU	International Telecommunication Union
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	HTTP over SSL
ICMP	Internet Control Message Protocol
IGMP	Internet Group Management Protocol
ILEC	Incumbent Local Exchange Carrier
IP	Internet Protocol
ISP	Internet Service Provider
ITSP	Internet Telephony Service Provider
IVR	Interactive Voice Response
LAN	Local Area Network
LBR	Low Bit Rate
LBRC	Low Bit Rate Codec
MC	Mini-Certificate
MGCP	Media Gateway Control Protocol

MOH	Music On Hold
MOS	Mean Opinion Score (1-5, the higher the better)
ms	Millisecond
MSA	Music Source Adaptor
MWI	Message Waiting Indication
OSI	Open Switching Interval
PCB	Printed Circuit Board
PR	Polarity Reversal
PS	Provisioning Server
PSQM	Perceptual Speech Quality Measurement (1-5, the lower the better)
PSTN	Public Switched Telephone Network
NAT	Network Address Translation
OOB	Out-of-band
REQT	(SIP) Request Message
RESP	(SIP) Response Message
RSC	(SIP) Response Status Code, such as 404, 302, 600
RTP	Real Time Protocol
RTT	Round Trip Time
SAS	Streaming Audio Server
SDP	Session Description Protocol
SDRAM	Synchronous DRAM
sec	seconds
SIP	Session Initiation Protocol
SLA	Shared line appearance
SLIC	Subscriber Line Interface Circuit

SP	Service Provider
SSL	Secure Socket Layer
TFTP	Trivial File Transfer Protocol
TCP	Transmission Control Protocol
UA	User Agent
uC	Micro-controller
UDP	User Datagram Protocol
URL	Uniform Resource Locator
VM	Voicemail
VMWI	Visual Message Waiting Indication/Indicator
VQ	Voice Quality
WAN	Wide Area Network
XML	Extensible Markup Language

C

Where to Go From Here

Cisco provides a wide range of resources to help you and your customer obtain the full benefits of the Cisco Small Business IP Telephony Device.

Product Resources

Resource	Location
Technical Documentation	Voice System (SPA9000 and SPA400): www.cisco.com/en/US/products/ps10030/tsd_products_support_series_home.html Voice Gateways/Analog Telephone Adapters: www.cisco.com/en/US/products/ps10024/tsd_products_support_series_home.html IP Phones: www.cisco.com/en/US/products/ps10033/tsd_products_support_series_home.html
Firmware Downloads	Go to tools.cisco.com/support/downloads, and enter the model number in the Software Search box.
Cisco Community Central > Small Business Support Community	www.myciscocommunity.com/community/smallbizsupport/voiceandconferencing
Phone Support	www.cisco.com/en/US/support/tsd_cisco_small_business_support_center_contacts.html
Warranty and End User License Agreement	www.cisco.com/go/warranty

Resource	Location
Open Source License Notices	www.cisco.com/go/osln
Regulatory Compliance and Safety Information	See the Technical Documentation pages listed above.
Cisco Partner Central site for Small Business	www.cisco.com/web/partners/sell/smb
Cisco Small Business Home	www.cisco.com/smb