DEPLOYMENT GUIDE

6.3.1 | February 2019 | 3725-85481-002A

Polycom® OBi Edition Deployment Guide

6001 America Center Drive
San Jose, CA 95002
USA

Disclaimer While Polycom uses reasonable efforts to include accurate and up-to-date information in this document, Polycom makes no warranties or representations as to its accuracy. Polycom assumes no liability or responsibility for any typographical or other errors or omissions in the content of this document.

Limitation of Liability Polycom and/or its respective suppliers make no representations about the suitability of the information contained in this document for any purpose. Information is provided "as is" without warranty of any kind and is subject to change without notice. The entire risk arising out of its use remains with the recipient. In no event shall Polycom and/or its respective suppliers be liable for any direct, consequential, incidental, special, punitive or other damages whatsoever (including without limitation, damages for loss of business profits, business interruption, or loss of business information), even if Polycom has been advised of the possibility of such damages.

End User License Agreement BY USING THIS PRODUCT, YOU ARE AGREEING TO THE TERMS OF THE END USER LICENSE AGREEMENT (EULA) AT: http://documents.polycom.com/indexes/licenses. IF YOU DO NOT AGREE TO THE TERMS OF THE EULA, DO NOT USE THE PRODUCT, AND YOU MAY RETURN IT IN THE ORIGINAL PACKAGING TO THE SELLER FROM WHOM YOU PURCHASED THE PRODUCT.

Patent Information The accompanying product may be protected by one or more U.S. and foreign patents and/or pending patent applications held by Polycom, Inc.

Open Source Software Used in this Product This product may contain open source software. You may receive the open source software from Polycom up to three (3) years after the distribution date of the applicable product or software at a charge not greater than the cost to Polycom of shipping or distributing the software to you. To receive software information, as well as the open source software code used in this product, contact Polycom by email at OpenSourceVideo@polycom.com.

Customer Feedback We are striving to improve our documentation quality and we appreciate your feedback. Email your opinions and comments to DocumentationFeedback@polycom.com.

Polycom Support Visit the Polycom Support for End User License Agreements, software downloads, product documents, product licenses, troubleshooting tips, service requests, and more.

Before You Begin

This guide describes how to develop and deploy XML-based applications to configure Polycom® VVX® Business IP Phones, OBi Edition.

The information applies to the following OBi Edition phones:

- VVX 150, OBi Edition
- VVX 250, OBi Edition
- VVX 350, OBi Edition
- VVX 450, OBi Edition

Audience, Purpose, and Required Skills

This guide is written for a technical audience involved in developing and deploying XML-based applications for OBi Edition phones. This audience includes Internet Telephony Service Providers (ITSP), Managed Service Value Added Resellers (VAR), Internet Telephony Professionals, and Technology Hobbyists.

You must be familiar with the following concepts before beginning:

- Current telecommunications practices, protocols, and principles.
- Telecommunication basics, video teleconferencing, and voice or data equipment.
- OpenSIP networks and VoIP endpoint environments.

Note for Australian readers: Throughout this document, we refer to ITSP – treat this term the same as you would for VSP (Voice Service Provider).

This document targets the following audiences:

- Service providers who to deploy and remotely manage Polycom® VVX® business IP phones, OBi Edition, using a central provisioning system.
- VARs planning to support customers remotely, managing OBi Edition phones via a central provisioning system.
- Power users of OBi Edition phones who want to remotely manage their phones for their friends and family.

Note to End Users

This guide targets phone administrators who manage large installed bases.

End users are highly encouraged to use the OBiTALK web portal at www.obitalk.com to configure and manage their OBi devices, or to perform management tasks locally by using the phone's native web portal, accessible by logging into the phone from any PC with a web browser. See the *Polycom® VVX® Business IP Phones, OBi Edition User Guide* for more information about your phone's native web portal.

Get Help

For more information about installing, configuring, and administering Polycom products, see **Documents & Software** at https://support.polycom.com.

Polycom and Partner Resources

Polycom has a number of options available to customers who need help with their Polycom OBi Edition products. In addition to this guide, the following documents and other resources provide more details:

- Visit the OBi Edition documents and guides found at https://documents.polycom.com
  - *Polycom® VVX® Business IP Phones, OBi Edition Administration Guide*
  - *Polycom® OBi Edition Provisioning Guide*
  - *Polycom VVX Business IP Phones, OBi Edition User Guide*
- For Polycom UC Software releases and documentation, see Polycom Voice Support.
- For user guides for Polycom voice products, refer to the product support page for your phone at Polycom Voice Support.
- For help or technical support for your phones, you can search for Polycom documentation at the Polycom Unified Communications (UC) Software Resource Center.
- You can find Request for Comments (RFC) documents by entering the RFC number at http://www.ietf.org/rfc.html.
- Email the Polycom OBi Edition Service Provider Support Team at: Obi.SPSupport@Polycom.com

To find all Polycom partner solutions, see Strategic Global Partner Solutions.

The Polycom Community

The Polycom Community gives you access to the latest developer and support information. Participate in discussion forums to share ideas and solve problems with your colleagues. To register with the Polycom Community, simply create a Polycom Online account. When logged in, you can access Polycom support personnel and participate in developer and support forums to find the latest information on hardware, software, and partner solutions topics.

Documentation Feedback

We welcome your feedback to improve the quality of Polycom documentation.

You can email Documentation Feedback for any important queries or suggestions related to this documentation.

Notational Conventions

This guide provides device configuration parameters and their values in the following formats:

- Canonical fashion
- Literal fashion

Both notational conventions point to the same parameters, but their appearances are different.

The canonical fashion simplifies locating parameters on the phone's native web portal or on OBiTALK.com.

The literal fashion is required when provisioning or writing OBIPhoneXML apps.

Canonical Fashion

This example shows the format of the canonical fashion.

- ***Parameter Group Name***::**ParameterName** = `Parameter Value` {replace with actual value}

The ***Parameter Group Name*** is the heading of the parameter group on the left side panel of the device local configuration or OBiTALK Configuration web page. This string may contain spaces. When a group heading has more than one level, each level is separated with a –, such as:

- ***Services Providers - ITSP Profile A – SIP***:

The **ParameterName** is the name of the parameter as shown on the web page and MUST NOT CONTAIN ANY SPACES. ***Parameter Group Name*** and **ParameterName** are separated by two colons (::), as shown in the first example above.

The `Parameter Value` is the literal value to assign to the named parameter and may contain spaces. You can omit ***Parameter Group Name*** or its top-level headings when the context is clear. For example:

- ***SP1 Service***::**AuthUserName** = `4082224312`
- ***ITSP Profile A - SIP***::**ProxyServer** = `sip.myserviceprovider.com`
- **ProxyServerPort** = `5082`

Literal Fashion

These examples show the format of the literal fashion. The literal fashion is used when provisioning or writing OBIPhoneXML apps.

- ***ParameterGroupName***.**ParameterName**.`Parameter Value` {replace-with-actual-value}
- ***Parameter.Group.Name.ParameterGroupName.*****ParameterName.**`Parameter Value`

The ***ParameterGroupName.*** is the name of the first parameter group in literal fashion. This string MUST NOT CONTAIN ANY SPACES, and always is terminated with a period, as shown. More than one ***ParameterGroupName.*** may be used. The ***ParameterGroupName.*** is case-sensitive.

The **ParameterName.** is the name of the parameter, and always is terminated with a period, as shown. This string MUST NOT CONTAIN ANY SPACES. The **ParameterName.** is case-sensitive.

The `Parameter Value` is the literal value to assign to the named parameter and may contain spaces. The `Parameter Value` is not case-sensitive, but it MUST EXACTLY MATCH the value when one or more choices are available.

When using the literal fashion in your XML, you need to exactly match the text string for ***ParameterGroupName***.**ParameterName**.`Parameter Value`, but text formatting such as bold face is not required and will be removed when your script or app is processed.

Boolean Values

You can identify parameters that take a Boolean value on your phone's configuration web pages by a check box next to the parameter name. Throughout the document, we may loosely refer to a Boolean value as "enable/disable" or "yes/no", but the only valid Boolean parameter values to use in a phone configuration file is either `true/false` or `True/False` (case-sensitive). This is equivalent to selecting or clearing the check box on the configuration web pages.

Multiple Choice Values

You must provision parameters that take one of several valid options from a drop-down list on the device message with string values that match exactly one of those choices. Otherwise, the device uses the default choice. Matching the provisioned value against valid strings is case-sensitive and doesn't allow extra spaces.

Parameter Values

When entering a parameter value from the web page or via provisioning, avoid adding extra white spaces before or after the parameter value. If the value is a comma-separated list of strings or contains attributes after a comma or semicolon, avoid adding extra white space before and after the delimiter.

For example: **CertainParameter** `= 1,2,3,4;a;b;c`

If a parameter value can include white spaces, such as **X_DisplayLabel**, use just a single space and no extra space before and after the value.

For example: **X_DisplayLabel** `= My New Service`

Getting Started

The OBi Edition VVX business IP phones provide a programmatic interface for developing third-party applications that you can download and execute on the phones. You can develop these applications (called OBiPhoneXML apps) using a proprietary mark-up language called OBiPhoneXML to create solutions involving deployed phones and servers.

Typical applications include:

- Access to corporate directories
- Voicemail navigation
- Call queue monitoring
- Integration with customer databases

XML App Execution Models

An XML app comprises one or more scripts written in OBiPhoneXML that you download and execute sequentially on the phone to achieve the desired behavior. The execution of an app starts with the first XML script, also known as the *landing script*. Depending on the interaction with the user and relevant events happening on the phone, the phone can invoke and execute additional scripts. The server that generates or stores the scripts is the (XML) App Server.

The phone uses two models to execute an XML app:

- The *pull model* is where the phone pulls the landing script from a preconfigured action URL triggered by some event on the phone (such as user pressing a feature key). The method supported by the phone for pulling is `HTTP/HTTPS GET`.
- The *push model* is where the App Server pushes the script to the phone for execution. The method supported by the phone for pushing is `SIP NOTIFY` with the Event: `obihai-xml` and Content-Type: `application/xml`.

Action URLs – Pull Model

The OBi Edition phones support the pull model by accepting the configuration of the app URL into the phone to associate with a softkey or a feature key, such that when the key is pressed, the phone downloads the corresponding script from the given URL, applies the script, and executes the script. This URL is called an Action URL.

Action URL Feature Key

To invoke an Action URL with a feature key, the feature must be set up with the function `Action URL`, and the URL of the landing script must be specified in the `Number` field of that feature key. The `Name` field of the

feature may also be configured with a friendly name to be displayed on screen to identify the functionality of the script. For example:

```
Function = Action URL
Number =
http://xmlapp-server.obihai.com:8080/xml/testmain.xml?user=jsmith&model=1
032
Name = Main Menu
```

Action URL Softkey

To invoke an Action URL with a softkey, include a softkey with the id `acturl` in a softkey set parameter, with the URL specified in a `url` attribute. For example, the default Home softkey set is: `redial, cfa,dnd,missed|lines`. You can replace the fourth softkey with an Action URL (all one line):

```
redial,cfa,dnd,acturl;url="http://10.1.1.123/test.xml?user=abc&amp;model=1032"
;label="Main Menu"
```

Note that the value of the `url` attribute MUST be XML-escaped. For example, `&` must be specified as `&`.

SIP NOTIFY – Push Model

As it is typically not feasible for the App Server to send HTTP requests to the phone, the phone only accepts server-pushed XML via SIP NOTIFY requests sent to it over one of the enabled SP*n* service channels. The SIP NOTIFY must have the Event: `obihai-xml` with the Content-Type: `application/xml`.

The phone also accepts XML pushed via HTTP POST sent to the built-in Web Server port with the following URL: `http://{device-ip-address}/obihai-xml` with the Content-Type: `application/xml`.

When the XML pushed to the phone contains one or more `<ExecuteItem>` elements, the phone replies to the request with the inclusion of a message body that is an `<ObihaiIPPhoneExecuteResponse>` containing one or more `<ExecuteItem>` elements in the same order as the corresponding `<ExecuteItem>` elements in the original request. The `<ExecuteItem>` element in the response contains a result attribute that indicates the status of executing the item. It may contain additional information as a result of carrying out the `<ExecuteItem>` element.

For example, when the following XML is pushed to the phone to make a call:

```
<ObihaiIPPhoneExecute>
  <ExecuteItem id="1" URI="Dial:14089991234"/>
</ObihaiIPPhoneExecute>
```

The following message body may be included with the response returned by the phone:

```
<ObihaiIPPhoneExecuteResponse>
  <ExecuteItem id="1" result="200 OK" call-id="f234ab97"/>
</ObihaiIPPhoneExecuteResponse>
```

Event-Driven Model

The phone is programmed to perform specific actions in response to a prescribed set of events.

Events consist of:

- User actions (such as a key press)
- Telephony events (such as an incoming call)
- Server-to-phone push messages (via SIP NOTIFY Events)
- Programmed timer expirations

Actions consist of:

- Activating internal phone features
- Playing audio files and tones
- Phone screen updates
- Phone-to-server pull requests (via the HTTP protocol)

The phone configuration profile specifies the initial state and event triggering hooks for each application (such as through softkey configuration).

Typical Application Event Flow

In a typical application, a feature key is configured with an application-specific URI. When you press that key, the phone activates the URI, sending an HTTP GET request to an application server.

The server response to this request consists of an XML page, whose contents specify both visual content for the phone to display, and also actions to perform under various triggers. In this way, the phone display acts as a kind of browser on behalf of the application. User actions or other events then trigger the phone to perform a prescribed action, and possibly generate additional server requests for updated XML content.

Thus, the phone-side of the application design revolves around the XML content of messages (pages) sent by the server to the phone. These are either synchronous responses to phone HTTP requests, or else messages pushed by the server to the phone asynchronously (via SIP NOTIFY Events).

Configuring Keys to Launch Phone Apps

You can preconfigure the phone to map line keys and programmable keys to phone applications. This enables a phone user to run an application by pressing the specified key. For this use case, the key is configured with the URL from which to fetch the topmost page of the application.

Map a Line Key to a Phone App

Configure the following parameters to set `LineKey{n}` as the trigger for running an application:

Configuration Parameters for a Line Key

Configuration Parameter	Programmed Value
VoiceService.1.Phone.LineKey.{n}.Function	“Action URL”

Configuration Parameters for a Line Key

Configuration Parameter	Programmed Value
VoiceService.1.Phone.LineKey.{n}.Name	Application display name
VoiceService.1.Phone.LineKey.{n}.Number	URL from which to fetch the application's topmost XML page. The phone generates an HTTP GET request to the server when the key is pressed.

Map a Programmable Key to a Phone App

Alternatively, if a programmable key is desired as the trigger, the corresponding "ProgKey" entries would be configured in place of "LineKey".

Upon receiving the XML page in response to the HTTP GET, the phone interprets the contents as described in the following sections.

Map a Softkey to a Phone App

You can also configure a softkey to invoke an application, with a similar syntax as in the following example, inserted in a softkey set parameter (under the Soft Keys configuration page):

```
acturl;url=http://192.168.15.225:8080/obixml/testmain.xml;label="Local Tests"
```

Introduction

Before you use your OBi Edition phone, take a few moments to familiarize yourself with its features and user interface.

The terms “the phone” and “your phone” refer to any of the OBi Edition VVX business IP phones. Unless specifically noted in this guide, all phone models operate in similar ways.

As you read this guide, keep in mind that certain features are configurable by your system administrator or determined by your network environment. As a result, some features may not be enabled or may operate differently on your phone. Additionally, the examples and graphics in this guide may not directly reflect what is displayed or is available on your phone screen.

The OBi Edition phones share the same functionalities with all other Polycom business IP phones:

- Support for all standard SIP-based IP PBX and ITSPs/VSPs.
- Support for 3CX PBX and uaCSTA interoperability.
- Suited for all service provider and enterprise deployment environments, regardless of size.
- Ideal for self-service installations. Home users, small business owners, or corporate IT departments can easily install, set up, and manage these phones.
- Seamless integration with popular softswitch architectures
- Cloud management enabled via OBiTALK.com with both a user portal and an ITSP partner portal with an optional REST API.

An area available only to service providers, the https://www1.OBiTALK.com ITSP portal may also be used by service providers for phone provisioning, management, and troubleshooting. The OBiTALK ITSP portal can be used independently as the sole system for secure management of OBi Edition phones or in conjunction with an existing centralized provisioning system managed by the service provider.

By design, all OBi edition phones are capable of remote management by a service provider. You can update your firmware remotely to provide new features and services. You can update phone configuration to handle user requests and service enhancements. You can remotely monitor your phones for troubleshooting and routine health check-ups.

This document describes the technologies and methods to manage these phones remotely and to securely provision your phones at a massive scale. A complete listing of available configuration parameters on the phone is given at the end of this document. For a complete phone parameter reference, please see the *Polycom VVX Business IP Phones, OBi Edition Administrator Guide*.

Phone Parameters and Objects

Every OBi Edition phone is a highly programmable device with more than a thousand configuration parameters. The configuration allows a user or service provider to control every aspect of its operation. Following the TR-104 standard naming convention, phone parameters are grouped into a small number of hierarchical objects. Each configuration parameter is identified by a unique canonical name composed of two parts: an object name and a parameter name. Parameters belonging to the same object share the same object name. Here is an example of a canonical name (**SP1 Service – Enable**):

- ***VoiceService.1.VoiceProfile.1.Line.1.*Enable**

where ***VoiceService.1.VoiceProfile.1.Line.1.***is the object name and **Enable** is the parameter name. Note that the object name must include the ending dot. Parameter names and object names are case-sensitive.

Each hierarchy of object is represented by a dot in the object name. When it is possible to have more than one instance of the same object, each instance is identified with an integral instance number starting with 1, 2, …, after the object name. For example, the **SP2/SP3/SP4/SP5/SP6 Service – Enable** parameters have the following literal names:

- ***VoiceService.1.VoiceProfile.1.Line.2.*Enable**
- ***VoiceService.1.VoiceProfile.1.Line.3.*Enable**
- ***VoiceService.1.VoiceProfile.1.Line.4.*Enable**
- ***VoiceService.1.VoiceProfile.1.Line.5.*Enable**
- ***VoiceService.1.VoiceProfile.1.Line.6.*Enable**

The above shows four instances of the ***VoiceService.1.VoiceProfile.1.Line.1.*** objects in the configuration under the ***VoiceService.1.VoiceProfile.1.*** object. Each ***Line.*** object instance corresponds to the parameters under one of the four SP services.

Here is another example using the **ProxyServer** parameter under the SIP section of ITSP Profile A, B, C, D, E, and F:

- ***VoiceService.1.VoiceProfile.1.SIP.* ProxyServer**
- ***VoiceService.1.VoiceProfile.2.SIP.* ProxyServer**
- ***VoiceService.1.VoiceProfile.3.SIP.* ProxyServer**
- ***VoiceService.1.VoiceProfile.4.SIP.* ProxyServer**
- ***VoiceService.1.VoiceProfile.5.SIP.* ProxyServer**
- ***VoiceService.1.VoiceProfile.6.SIP.* ProxyServer**

The above shows four instances of the ***VoiceService.1.VoiceProfile.*** objects, corresponding to ITSP Profile A, B, C, D, E, and F, respectively. Note that however that the ***Line.*** object is only defined under the ***VoiceService.1.VoiceProfile.1.*** object. It is undefined under the ***VoiceService.1.VoiceProfile.2.***, ***VoiceService.1.VoiceProfile.3.***, ***VoiceService.1.VoiceProfile.4.***, ***VoiceService.1.VoiceProfile.5.***, and ***VoiceService.1.VoiceProfile.6.*** objects. This helps reduce the total number of phone parameters.

Many of the objects and parameters are taken from the TR-104 standard with the same names, such as the ***VoiceService.1.VoiceProfile.1.Line.*** objects and the **ProxyServer** parameters shown earlier. There are many more objects and parameters that are not described in the TR-104 standard. For these objects and parameters, their names have the prefix ***X_*** attached to indicate that they are proprietary extensions. For example, there are eight instances of the ***VoiceService.1.X_VoiceGateway.*** objects, four instances of the ***VoiceService.1.X_TrunkGroup.*** objects, and a ***VoiceService.1.VoiceProfile.1.Line.1.*X_RingProfile** parameter. Note that if the object name has the ***X_*** prefix, no ***X_*** prefix is needed in the parameter name.

A notable special case is the ***SpeedDial***. object, which is proprietary and doesn't contain an instance number. It has 99 parameters in this object with the names 1, 2, 3, … 99. Hence the parameter names are **SpeedDial.1**, **SpeedDial.2**, … **SpeedDial.99**, which must not be misinterpreted as 99 instances of the ***SpeedDial.*** object.

For convenience, we may exclude the object name when referring to a parameter in this document when the context is clear. For example, we may simply refer to **ConfigURL** without mentioning its object name ***X_DeviceManagement.Provisioning.***

The Object Name is Just a Name

It should be emphasized that the use of TR-104 object names is simply to divide the parameter naming space such that the phones can be referenced and managed more conveniently. In general, all objects in the phone configuration should be assumed to be independent of each other, in the sense that they do not inherit any properties from their "parent" despite their names are children of another object in syntax. Sibling objects in this sense also do not share any common properties.

For example, the parameters in the object ***VoiceService.1.VoiceProfile.1.Line.1.*** (parameters under SP1 Service on the phone web page) have no particular relationship to the parameters in the object ***VoiceService.1.VoiceProfile.1.*** object (parameters under **ITSP Profile A–General** on the phone web page). You can set up an ITSP account on SP1 Service that refers to any of the available ITSP Profiles.

How the Phone Organizes SP Account Parameters

The best way to understand how parameters are organized in your phone is by studying the parameter layout on the device web pages. A service provider user account is primarily configured under an SP*n* Service menu on the device web page, where *n* = 1 through 6. There you can configure the **AuthUserName** and **AuthPasssword** parameters of each user account. This task is similar to the user-id and password parameters found in similar products, among other relevant information. Each SP service contains a parameter that points to an ITSP*x* profile, where *x* = A through F. An ITSP profile is where you configure parameters specific to the SP but not specific to individual user accounts.

The **ProxyServer** and **RegistrationPeriod** parameters are examples of such SP-specific parameters. With this organization, a device with two user accounts from the same ITSP can be configured on two different SP*n* services that refer to the same ITSP*x* profile. Following a similar strategy, SP*n* services contain parameters to point to a Tone Profile (A or B), a Ring Profile (A or B), and a Codec Profile (A or B). So two different SP*n* services on the same phone can share the same tone, ring, and codec definitions.

The following table shows the mapping from some SP account parameter objects to parameter groups on the device web page.

Sample SP Account Parameter Object Mappings

Provisioning Parameter Object	Parameter Group on Device Web Page	Notes
VoiceService.1.VoiceProfile.1.Line.n. (*n* = 1,2,3,4,5,6)	**Voice Services/SP*n* Service – SP*n* Service** (*n* = 1,2,3,4,5,6)	These three objects (with the same object instance number *n*) completely define an SP*n* Service on the web page.
VoiceService.1.VoiceProfile.1.Line.n.SIP. (*n* = 1,2,3,4,5,6)	**Voice Services/SP*n* Service – SIP Credentials** (*n* = 1,2,3,4,5,6)	
VoiceService.1.VoiceProfile.1.Line.n.CallingFeatures. (*n* = 1,2,3,4,5,6)	**Voice Services/SP*n* Service – Calling Features** (*n* = 1,2,3,4,5,6)	
VoiceService.1.VoiceProfile.n. (n = 1,2,3,4,5,6)	**ITSP Profile x/General – General** (*x* = A,B,C,D corr. *n* = 1,2,3,4,5,6)	These four objects (with the same object instance number *n*) together completely define an ITSP Profile *x* on the web page. When an SP Service refers to ITSP Profile *x*, it's referring to the four objects as a group. The SP Service parameter **X_ServProvProfile** binds the SP service to the ITSP profile.
VoiceService.1.VoiceProfile.n.ServiceProviderInfo. (*n* = 1,2,3,4,5,6)	**ITSP Profile x/General – Service Provider Info** (*x* = A,B,C,D,E,F corr. *n* = 1,2,3,4,5,6)	
VoiceService.1.VoiceProfile.n.SIP. (*n* = 1,2,3,4,5,6)	**ITSP Profile x/SIP – SIP** (*x* = A,B,C,D,E,F corr. *n* = 1,2,3,4),5,6	
VoiceService.1.VoiceProfile.n.RTP. (*n* = 1,2,3,4,5,6)	**ITSP Profile *x*/RTP – RTP** (*x* = A,B,C,D,E,F corr. *n* = 1,2,3,4,5,6)	
VoiceService.1.VoiceProfile.1.Line.n.Codec. (*n* = 1,2)	**Codecs/Codec Profile *x*** (*x* = A,B corr. *n* = 1,2)	The SP Service parameter **X_CodecProfile** binds the SP service to the Codec Profile.
VoiceService.1.VoiceProfile.1.Line.n.Ringer. (*n* = 1,2)	**Ring Settings/Ring Profile *x*** (*x* = A,B corr. *n* = 1,2)	The SP Service parameter **X_RingProfile** binds the SP service to the Ring Profile.
VoiceService.1.Phone.	**Physical Interfaces/Phone**	Your phone isn't hardwired to any SP service. It can use any service to make a call. Incoming calls on any SP service can be directed to ring the phone.
VoiceService.1.X_StarCode.n. (*n* = 1,2)	**Star Codes/Star Code Profile *x*** (*x* = A,B corr. *n* = 1,2)	Your phone has the parameter **StarCodeProfile** to bind a Star Code profile.
VoiceService.1.VoiceProfile.n.Tone. (*n* = 1,2)	**Tone Settings/Tone Profile x** (*x* = A,B corr. *n* = 1,2)	Your phone has the **ToneProfile** parameter to bind to a Tone Profile.
SpeedDial.	**User Settings/Speed Dials**	Speed Dials are shared among all lines on the phone.

Your phone isn't necessarily bound to just one of the SP Services configured on the phone. The SP Services are completely decoupled from the lines. By default, you can make calls to any of the SP Services from any line, and incoming calls on any SP Service are set to ring all the lines. On the other hand, the device

configuration is flexible enough to mimic the legacy behavior of hard-wiring each line to a different SP Service, if it's necessary to have such restriction. The binding of line to SP service can be manipulated using a combination of the parameters listed in the following table. Refer to the *Polycom VVX Business IP Phones, OBi Edition Administrator Guide* for details on using these parameters.

Each instance of SP Services, ITSP Profiles, Codec Profiles, Ring Profiles, and Tone Profiles is independent. The instances of the same objects do not share common properties. This means you can use completely different SIP and RTP configurations for two different accounts, or completely different gain, impedance, hook flash timings, and caller ID settings for each line. You have complete flexibility when it comes to configuring multiple accounts on the device. At the same time, if two accounts on the same device share the same characteristics, you can set up the two SP services to point to the same instance of the objects that define those common characteristics, like an ITSP Profile or Ring Profile. Hence, you don't need to define the same parameters for the object more than once, saving time and space.

Which Objects to Configure

By now, you should have a pretty good idea of how configuration parameters are organized in your phone. If you only need to configure one account on the phone for the service you offer, select an available SP Service slot (say SP1) and an available ITSP Profile slot (say ITSP Profile B), and configure the ITSP-specific information and user-specific information on those objects accordingly. In particular, the SP 1 Service you selected must have its ***X_ServProvProfile*** pointing to ITSP Profile B.

If you want to configure two accounts on your phone, you must select a different SP Service slot for each account (say SP1 and SP2). Now you have the choice of using just one ITSP Profile for both accounts, or have a different profile for each. The choice is simple: If the parameters in the ITSP profile can be set to the same for both accounts, then using the same ITSP profile for both is more efficient and convenient. But if at least one parameter must be different, such as the **DigitMap** (under **ITSP Profile x/General** on the phone web page), you need to use a different ITSP Profile for each SP account. Similar comments can be made regarding Tone Profile, Ring Profile, and Codec Profile.

Macros

The OBi Edition enables you to use macros when provisioning your phones, which speeds and simplifies the process.

Parameter Macro Expansion

You may specify parts of or the entire value of a parameter with parameter macros. A parameter macro has the general format $NAME or ${NAME}, where NAME is the name of a defined macro. Macro names are case-sensitive. The curly braces `{}` are optional except when the name is followed by a character in the set [a-zA-Z0-9]. For example, the macro $MAC represents the MAC address of the current device, and it can be used as part of a parameter value, such as:

```
ConfigURL = http://ps.abc.com/obi${MAC}.xml
```

The macro is expanded by the device with the actual value it represents when the parameter value is loaded. If the macro name is undefined, the macro name is used as-is, including the $ and any enclosing braces.

Macros help to keep the device profile more generic so that the same profile may be applied to all units. Note that some macros may be used in specific parameters only, while others may be used in all parameters.

The following table lists the macros currently defined with the given properties, where:

- Value – The value in which the macro is expanded.
- ExpandIn – The parameter in which the macro can be used – ANY means it can be used in any parameter.
- Script – Whether the value of the macro can be changed when used in a Provisioning Script (**ConfigURL**).
- Web – Whether the value of the macro is shown on the device web page.
- Provisioning – Whether the value of the macro can be changed by provisioning.

Currently Defined Macros

Macro Name	Value	ExpandIn	Script	Web	Provisioning
MAC	MAC address in uppercase, such as 9CADEF000000	ANY	N	Y	N
MACC	MAC address in uppercase with colons, such as 9C:AD:EF:00:00:00	ANY	N	N	N
mac	MAC address in lowercase, such as 9cadef000000	ANY	N	N	N
macc	MAC address in lowercase with colons, such as 9c:ad:ef:00:00:00	ANY	N	N	N
FWV	F/W version, such as 6.3.0.15058	ANY	N	Y	N
HWV	H/W version, such as 1.0	ANY	N	Y	N
IPA	Device IP Address, such as 192.168.15.100	ANY	N	Y	N
DM	Device Model Name, such as VVX450	ANY	N	Y	N
DMN	Device Model Number, such as 450	ANY	N	Y	N
OBN	Device OBi Number, such as 723822495	ANY	N	Y	N
DSN	Device S/N, such as 64167F3A539F	ANY	N	Y	N
DHCPOPT66	Option 66 offered by the DHCP server	ANY	N	N	N
SPRM0 to SPRM7	**X_DeviceManagement.ITSPProvisioning.SPRM0** to **X_DeviceManagement.ITSPProvisioning.SPRM7**	**X_DeviceManagement.ITSPProvisioning.ConfigURL** and **X_DeviceManagement.FirmwareUpdate.FirmwareURL**	Y	N	Y

Currently Defined Macros

Macro Name	Value	ExpandIn	Script	Web	Provisioning
GPRM0 to GPRM7	**X_DeviceManagement.ITSPProvisioning.GPRM0** to **X_DeviceManagement.ITSPProvisioning.GPRM7**	**X_DeviceManagement.ITSPProvisioning.ConfigURL** and **X_DeviceManagement.FirmwareUpdate.FirmwareURL**	Y	Y	Y
TPRM0 to TPRM3	**X_DeviceManagement.ITSPProvisioning.TPRM0** to **X_DeviceManagement.ITSPProvisioning.TPRM3**	**X_DeviceManagement.ITSPProvisioning.ConfigURL** and **X_DeviceManagement.FirmwareUpdate.FirmwareURL**	Y	Y	Y
UDM0 to UDM3	**X_DeviceManagement.X_UserDefineMacro.0.Value** to **X_DeviceManagement.X_UserDefineMacro.3.Value**	**X_DeviceManagement.X_UserDefineMacro.0.ExpandIn** to **X_DeviceManagement.X_UserDefineMacro.3.ExpandIn**	Y	Y	Y
UDM4 to UDM31	**X_DeviceManagement.X_UserDefineMacro.4.Value** to **X_DeviceManagement.X_UserDefineMacro.31.Value**	**X_DeviceManagement.X_UserDefineMacro.4.ExpandIn** to **X_DeviceManagement.X_UserDefineMacro.4.ExpandIn**	Y	N	Y

User-Defined Macros

In addition to the predefined macros, the configuration can specify as many as 32 user-defined macros. These macros are named `$UDM0`, `$UDM1`, `$UDM2`, …, `$UDM31`. Only `$UDM0` to `$UDM3` are accessible from the device web page. The rest are hidden, and can be changed only by provisioning. To define a user macro, specify its properties in the corresponding object parameters as shown in the following table:

User-Defined Macros

UDMx Parameters, x = 0, 1, 2, …, 3	Description
X_DeviceManagement. X_UserDefineMacro.x.Value	The value can be any plain text or a valid canonical parameter name preceded by a $ sign. For example: `$X_DeviceManagement.WebServer.Port` Note: Here you MUST NOT enclose the parameter name following the $ sign with braces or parentheses.
X_DeviceManagement. X_UserDefineMacro.x.ExpandIn	This is a comma-separated list of canonical parameter names, where the macro expansion can be used. As many as three parameter names may be specified. Specify ANY to allow the macro to expand in any parameter. Example: `X_DeviceManagement.HTTPClient.UserAgent` Note: There is no $ sign in front of the parameter name. The macros may not be used in any parameter value if this value is set to blank (the default).

As an example, for the phone to request configuration from a provisioning server using HTTP, and for this HTTP request to include a request parameter that is a 4-digit code stored in Speed Dial #99 by a new subscriber, you can set up `$UDM0` for this according to the following table:

$UDM0 Settings

Parameter Name	Value
X_DeviceManagement.X_UserDefineMacro.0.Value	**$SpeedDial.99**
X_DeviceManagement.X_UserDefineMacro.0.ExpandIn	**X_DeviceManagement.ITSPProvisioning.ConfigURL**
X_DeviceManagement.ITSPProvisioning.ConfigURL	`http://prov.myitsp.com/obi${mac}-signup.xml?code=${UDM0}`

Note: The new subscriber may enter a random code, say `8714`, ZTP into speed dial 99 by dialing the following star code sequence from a connected phone: `*74 99 8714#`. The subscriber may find out information about this process on the ITSP's web site that also generates the 4-digit random code to be stored in Speed Dial 99. The example parameter shown here may be preinstalled in the phone as part of its ZTP profile. Subsequent provisioning of the device may clear the Speed Dial to prepare for normal usage by the subscriber.

OBi Edition Phone Provisioning

Your phones are highly programmable devices with more than a thousand configuration parameters on each. Configuration enables you to choose how and when you provision your phones with these parameters. You also can choose how deeply to set system and user parameters, depending on the requirements of your system and your users.

Phone Configuration

Depending on your configuration needs, you can choose one or more of these phone configuration and provisioning methods:

- Remote phone configuration
- Local phone configuration
- Native web page configuration
- Polycom® Zero Touch Provisioning (ZTP) configuration

Remote Phone Configuration

The parameter set to upload to a deployed phone is stored in a device configuration file, also known as a device configuration profile, or simply *profile*. Profiles are served from a system known as the provisioning server that is usually managed by the service provider. A phone can be configured to pull its latest profile from the server on each reboot, and then periodically at regular intervals (once per day, for instance). This method of provisioning a phone is referred to as remote provisioning.

The URL for the phone to download a profile is specified in a parameter named **ConfigURL**. In its most basic form, the parameter is a standard URL of the profile, such as:

```
https://myiptsp.com/obi-092b3c003412.cfg
```

The full syntax of **ConfigURL** is a provisioning script that allows you to specify additional attributes such as the crypto and the encryption key and error handling. For a full description of the **ConfigURL** parameter, see the Provisioning Scripts section in this document, or refer to the *Polycom VVX Business IP Phones, OBi Edition Administrator Guide*.

To provide a plug-and-play User Experience, the service provider should at least configure the **ConfigURL** parameter before shipping phones to end users. It would appear that the SP must therefore touch each phone to insert this step and repackage the phone before shipping. Ideally, this step may be eliminated if the phones can be customized for the service provider at the factory or via remote customization. A customization service, known as Polycom® Zero Touch Provisioning (ZTP), is available to serve this purpose. You can read more about it in the ZTP Phone Customization section.

You can use ZTP to change your phone's software from UCS to OBi Edition by loading the new firmware at boot time.

Local Phone Configuration

You can configure your phones locally by browsing the phone web pages from a web browser running on a computer. The prerequisites for accessing the native phone web page are:

- The phone is connected to the network with proper IP address assigned.
- A way to find out the current IP address of the phone. To do this, press the **Home** key, then select the **Product Info** app. The phone's IP address displays.

To log in to the native phone web page, browse to the URL `http://Device-IP-Address/`, where `Device-IP-Address` is the current IP address of the phone. You can view and change a phone's configuration as well as update its firmware on the native web page served locally from the phone. This method of phone configuration is referred to as local configuration or local device management.

The computer where the web browser runs on in this case is usually on the same LAN as the phone. Here, security is usually not a big concern as long as the LAN is secured from public 'hostile' networks. Obviously, this is not the preferred method for a service provider to manage a deployed phone. In fact, most service providers would rather disable this capability on the phone so that the end user cannot tamper with its configuration. However, a service provider may still use the phone web page in a lab environment when initially experimenting with the phone parameter settings for eventual locked-down remote mass-provisioning or to prepare a phone before it is shipped out to an end-user, then switch to remote provisioning after deploying the unit.

Access to the phone web pages may be protected by passwords. Two passwords can configured on the phone: an Admin Password and a User Password. If a nonempty value is configured for the corresponding password, a window displays to prompt the user to enter the user-id and password during the first visit. If the corresponding password is empty, however, the phone serves the pages without prompting for user-id and password.

These four parameters in the **X_DeviceManagement.WebServer.** object control the behavior of the phone's built-in web server:

Parameters That Control the Phone's Native Web Server

Parameter	Description
Port	The web server listen port. Default is 80, the standard HTTP port. Note: Setting this value to 0 disables all web server access.
AdminPassword	Admin login password. Default is `admin`.
UserPassword	User login password. Default is `user`.

If necessary, you can block end user access to the admin or user phone web pages by setting a nonempty password for both, but not reveal either password to the end user. However, it may be useful to allow the end-user access to a subset of the configuration parameters on the user web pages. For example, you could allow the end user to change the speed dials on the phone's user page. Via provisioning, the service

provider can specify the user permission on a parameter-by-parameter basis. The permission can be either read-only, read-write, or no-access (hidden from the web page). The profile syntax to set user access permission per parameter is found in the Device Configuration Profile Formats section.

Unlike configuration parameters, the functions under **System Management/Device Update** on the phone web page are not controllable via provisioning. For these functions, the following restrictions always apply when the current login is the user:

- **Firmware Update**: Removed so that user cannot update firmware or AA prompts.
- **Backup AA User Prompts**: Same as admin login.
- **Backup Configuration**: Backup parameters set with user read-only or read-write permission only.
- **Restore Configuration**: Restore parameters set with user read-write permission only.
- **Reset Configuration**: Reset parameters set with user read-write permission only.

In addition, you can register your phones on OBiTALK.com, which enables you to access all your phones without having to enter the URL `http://Device-IP-Address/` for each phone. The appearance between the OBiTALK web page and the native phone web page is different, but as an admin user you have identical access to settings.

ZTP Phone Customization

Polycom offers a phone customization service known as Zero Touch Provisioning (ZTP) that enables service provider customers to select the default values for phone network configuration parameters. Contact obi.spsupport@polycom.com for bootstrapping for SP deployment.

Factory Reset

A factory reset returns all parameters to the default values. Default values are either the customized default values or the values set by the currently installed firmware. All Call history is cleared as a result of factory reset.

To perform a factory reset from the phone UI's Main Menu, select **Settings**, and press the **Factory Reset** softkey. Press **OK** to confirm. Your phone prompts you to enter the admin password for this operation. The default admin password is `admin`. Also, a factory reset can be invoked via remote provisioning. The Device Configuration Profile Formats section describes this method.

End-User 'User' Parameter Space

As mentioned earlier, the service provider may allow end-user control of a subset of the phone parameters from the phone web page by specifying the user access attribute on a parameter-by-parameter basis via provisioning. The syntax is covered in the Device Configuration Profile Formats section. In addition, a user may change a parameter setting using a star code, such as *67. The service provider can decide which settings the user can access using star codes.

All user-changeable phone parameters constitute the user parameter space. Changes in the user parameter space are not reported back to the service provider. Therefore, the service provider must take care to exclude those parameters from the device profile so they don't overwrite the user changes. The service provider can choose to send down a special one-time profile when it is required to clear some user settings remotely.

Locking Parameters

A locked parameter is one that the end user isn't allowed to change on the phone web page. These include all parameters where the user read-write permission is set to either read-only or no-access. Each parameter has a default user read write permission (see the table at the end of this document). User read-write permission may be changed by provisioning only.

It is not enough to lock only the specific parameters that you want to hide from the user. A user-defined macro can be defined to point to any parameter, even the hidden ones. Therefore, the protection is more complete if all the user-defined macros are also locked, or at least limited to where those can be used.

Finally, to protect against user factory resetting hidden or read-only parameters to default values, set the **DeviceInfo.ProtectFactoryReset** parameter to 1. Refer to the Factory Reset section for a more in-depth discussion on factory reset.

Firmware Update

As with parameter configuration, you can update the phone firmware locally from the phone web page or the IVR. The service provider may also update the firmware remotely via provisioning.

You can use the following methods to update the firmware on your phones:

- Update from the phone web page
- Update from the IVR
- Update using the firmware URL parameter

Update From the Phone Web Page

The **System Management/Device Update** menu on the phone web page has a **Firmware Update** option that enables you to upload a firmware configuration file from your computer to your phone. This option is visible only if the current login is `admin`.

Your configuration file must be in XML format with the following information:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<PHONE_IMAGES>
   <REVISION >
      <PHONE_IMAGE>
         <VERSION>6.3.0.16863</VERSION> <!-- Example version number -->
             <PATH>http://192.168.1.1/path-to-the-folder</PATH>
             <!-- Example URL and path to folder containing the updates -->
      </PHONE_IMAGE>
   </REVISION>
</PHONE_IMAGES>
```

Update From the IVR

Select option 6 from the IVR main menu to see if new firmware is available from Polycom. If yes, follow the IVR instructions to start the update. This IVR option is protected by the IVR access password.

Update Using the FirmwareURL

Firmware update can be triggered via provisioning by setting up the **FirmwareURL** parameter with the URL to download the new firmware. The full syntax of **FirmwareURL** is a provisioning script that lets you specify things like error handling and retries. Refer to the Provisioning Scripts section for a full description of this parameter.

The URL of the firmware specified in the FirmwareURL parameter has the following format:

```
scheme://[userid:pwd@]hostname[:port]/path
```

where:

- `[…]` — Indicates that the enclosed syntax is optional.
- `scheme` — Must be TFTP, HTTP, or HTTPS.
- `userid:pwd` — User ID and password for Basic/Digest authentication (optional; used for HTT/HTTPS only).
- `hostname` — Hostname of the server hosting the firmware.
- `port` — Server port number (optional). If omitted, the default is port 69 for TFTP, port 80 for HTTP, or port 443 for HTTPS.
- `path` — Pathname to locate the firmware file on the server.

Example:

```
http://admin:password@www.myitsp.com/6-3-0-16863/3111-48450-001.sip.ld
```

Important Note:

- If the path contains the filename of the firmware file, the filename must be in the format shown in the above example.
- If the path doesn't contain the filename of the firmware file, the firmware automatically appends a filename in the format of `part-number.sip.ld` (`part-number` is the device part number).

The following table summarizes the parameters that control firmware update using **FirmwareURL**.

Parameters That Control Firmware Update Using FirmwareURL

Parameter	Description
X_DeviceManagement. FirmwareUpdate.Method	Choices are: • `Disabled` = Do not check for firmware upgrade from **FirmwareURL**. • `System Start` = Check for firmware upgrade from **FirmwareURL** on system start only. • `Periodically` = Check for firmware upgrade from **FirmwareURL** on system start and also periodically at the interval specified in the **Interval** parameter. • `Time of Day` = Check for firmware upgrade from **FirmwareURL** at certain time of day specified by the **TimeofDay** parameter. Note: The first firmware upgrade check on system start occurs after a random delay of 0 to 30 seconds.
X_DeviceManagement. FirmwareUpdate.Interval	When **Method** is set to `Periodically`, this parameter specifies the number of seconds between each checking of the firmware upgrade from **FirmwareURL**. If the value is 0, the phone checks just once on system start only (equivalent to setting **Method** to `System Start`).

Parameters That Control Firmware Update Using FirmwareURL

Parameter	Description
X_DeviceManagement. FirmwareUpdate.Timeof Day	Time of day in `hh:mm[+rr]` format, valid when method is set to `Time of Day`. Choices are: • `hh`: `0` to `23` • `mm`: `0` to `59` • `rr`: Maximum range of random delay in minutes (`0 to 360`). Set to `0` to cancel the random delay. If `rr` is omitted, the random delay is set to 30 minutes. Your phones always check once on system start, no matter what time it is. When NTP time is not available, your phones check for firmware updates every 60 minutes.

X_DeviceManagement. FirmwareUpdate. FirmwareURL	The basic syntax is a URL to download the firmware package. The full syntax is a provisioning script as described in the Provisioning Scripts section. The supported schemes are `http://, https://, a`nd `tftp://`. For example:
	`http://prov-server.myitsp.com/6-3-0-16863/3111-48450-001.sip.ld`
	or
	`http://$USR:$PWD@prov-server.myitsp.com/6-3-0-16863/`
	if authentication is required.
	Account information is stored in the `username` and `password` parameters.

X_DeviceManagement. FirmwareUpdate. DnsLookUp	DNS record to use when resolve the hostname used in **FirmwareURL**. Choices are: • `A Record Only` • `SVR Record Only` • `Try Both` Note: When choosing `Try Both`, the SVR record is adopted if both records are available.
X_DeviceManagement. FirmwareUpdate.DnsSrv Prefix	Specifies whether the prefix is inserted automatically when performing a lookup for SRV record. Choices are: • `NO prefix` • `With Prefix` • `Try Both` Note: • `http._tcp.` or `_tftp._udp.` is prepended to the name as a prefix, depending on the scheme used in **FirmwareURL**. • When choosing `Try Both`, the SVR record is adopted if both records are available.
X_DeviceManagement. FirmwareUpdate. Username	The account username if authentication is required when getting the firmware using http or https.
X_DeviceManagement. FirmwareUpdate. Password	The account password if authentication is required when getting the firmware using http or https.

Device Configuration Profile Formats

Device profiles for phone configuration are XML documents that can be pushed to or pulled by phones to properly provision them for use. This section describes their formats.

Device Profile Format

Device profiles for phone configuration are developed in the following formats:

- Full profile format
- Compact profile format

The format you choose depends on the provisioning you plan to perform on your phones.

Full Profile Format

A device profile is a simple two-level XML document with the root element `<ParameterList>` that encloses zero or more `<Object>` elements. Each `<Object>` element must include a single `<Name>` element followed by zero or more `<ParameterValueStruct>` elements. The `<Name>` element inside an `<Object>` element specifies the object's name, which must also include the ending dot. Each `<ParameterValueStruct>` specifies the name and value of a single parameter belonging to the enclosing object.

Below is a simplified schema of a full profile format configuration file.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema>
<xs:element name="ParameterList">
<xs:complexType>
  <xs:attribute name="X_Reset">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:pattern value="All|Voice|Router"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>

    <xs:element name="Object" maxOccurs="unbounded" minOccurs="0">
      <xs:complexType>
        <xs:element name="Name" type="xs:string"/>
          <xs:element name="ParameterValueStruct" maxOccurs="unbounded" minOccurs="0">
            <xs:complexType>
              <xs:element name="Name">
```

```
            <xs:complexType>
              <xs:simpleContent>
                <xs:attribute name="X_UserAccess" default="Default">
                  <xs:simpleType>
                    <xs:restriction base="xs:string">
                      <xs:pattern value="readOnly|readWrite|noAccess|Default"/>
                    </xs:restriction>
                  </xs:simpleType>
                </xs:attribute>
              </xs:simpleContent>
            </xs:complexType>
          </xs:element>

          <xs:element name="Value">
            <xs:complexType>
              <xs:simpleContent>
                <xs:attribute name="X_UseDefault" default="No">
                  <xs:simpleType>
                    <xs:restriction base="xs:string">
                      <xs:pattern value="Yes|No"/>
                    </xs:restriction>
                  </xs:simpleType>
                </xs:attribute>
              </xs:simpleContent>
            </xs:complexType>
          </xs:element>
        </xs:complexType>
      </xs:element>
    </xs:complexType>
  </xs:complexType>
</xs:element>
</xs:schema>
```

Referring to the above XML schema, the optional **X_Reset** attribute of the `<ParameterList>` element may take one of the following values:

- **All**: Factory reset all parameters
- **Voice**: Factory reset voice parameters only

If the parameter list contains any parameter objects, they're applied AFTER factory reset. For example:

```
<ParameterList X_Reset="All">
<!-- 0 or more parameter objects to follow -->
...
</ParameterList>
```

WARNING: X_Reset, if present in the profile, causes the phone to perform a full system reboot after it completes processed the profile. This attribute should be sent to the phone just once in a profile for the purpose of factory resetting all the parameters only.

- The optional **X_UserAccess** attribute of the `<Name>` element inside a `<ParameterValueStruct>` element may take of the following values:
  - `readOnly`: User can only read the parameter value from local phone web page
  - `readWrite`: User can only read and set the parameter value from local phone web page
  - `noAccess`: User can't see the parameter from local phone web page
  - `Default`: user read-write permission follows the default for that parameter
- This example profile sets the **ConfigURL** parameter to `"readOnly"` for user level access:

```
<Object>
<Name>X_DeviceManagement.Provisioning.</Name>
<ParameterValueStruct>
<Name X_UserAccess="readOnly">ConfigURL</Name>
<Value>http://prov-server.myitsp.com/obi${MAC}.xml</Value>
</ParameterValueStruct>
...
</Object>
```

- The optional **X_UseDefault** attribute of the `<Value>` element specifies whether to use the default value for that parameter. If a nonempty content is also specified for this element, the attribute value is ignored in favor of the given content

Remember that all the XML elements, attributes, names, and values in the configuration file are case-sensitive. The phone discards the file if it's malformed per XML standards. Any unrecognized elements and attributes are ignored. Any unrecognized parameter and object names are ignored also. Attributes with invalid values are ignored as if the attribute isn't present. An invalid parameter value is ignored and the value isn't applied (but an **X_UserAccess** attribute with valid values, if present, is still applied). Parameter values containing reserved XML characters, `>` (0x3E), `<` (0x3C), `&` (0x26), `&quot;` (0x22) and `&apos;` (0x27), must be properly escaped.

Here is an example of a valid profile with three objects:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- OBi Configuration File -->
<ParameterList>
  <Object>
    <Name>X_DeviceManagement.FirmwareUpdate.</Name>
      <ParameterValueStruct>
        <Name>Method</Name>
          <Value>System Start</Value>
        </ParameterValueStruct>
        <ParameterValueStruct>
          <Name>FirmwareURL</Name>
            <Value>
              IF ( $FWV &amp;lt;= 1.0.3.1890 ) FWU -T=TPRM2
               http://server.myinc.com/VVX250-1-1-0-1891.fw;
            </Value>
          </ParameterValueStruct>
        </Object>
        <Object>
          <Name>X_DeviceManagement.Provisioning.</Name>
            <ParameterValueStruct>
```

```
<Name>Method</Name>
  <Value>Periodically</Value>
</ParameterValueStruct>
<ParameterValueStruct>
  <Name>Interval</Name>
    <Value>3600</Value>
  </ParameterValueStruct>
  <ParameterValueStruct>
    <Name>ConfigURL</Name>
    <Value>SYNC  http://server.myinc.com/profile/$mac-init.cfg</Value>
  </ParameterValueStruct>
</Object>
<Object>
  <Name>DeviceInfo.WAN.</Name>
    <ParameterValueStruct>
      <Name>AddressingType</Name>
        <Value X_UseDefault="Yes"/>
      </ParameterValueStruct>
      <ParameterValueStruct>
        <Name>IPAddress</Name>
        <Value X_UseDefault="Yes"/>
      </ParameterValueStruct>
      <ParameterValueStruct>
        <Name>SubnetMask</Name>
        <Value X_UseDefault="Yes"/>
      </ParameterValueStruct>
      <ParameterValueStruct>
        <Name>DefaultGateway</Name>
        <Value X_UseDefault="Yes"/>
      </ParameterValueStruct>
      <ParameterValueStruct>
        <Name>DNSServer1</Name>
        <Value>192.168.15.18</Value>
      </ParameterValueStruct>
      <ParameterValueStruct>
        <Name>DNSServer2</Name>
        <Value>192.168.15.108</Value>
      </ParameterValueStruct>
    </Object>
  </ParameterList>
```

Compact Profile Format

The phone supports an alternative profile format that is more compact to reduce the file size of the profile. The element and attribute names in the full format have a corresponding short form as listed below:

- `<0> = <Object>`
- `<N> = <Name>`

- <V> = <Value>
- <P> = <ParameterValueStruct>
- <X_R> = X_Reset
- <X_UD> = X_UseDefault
- <X_UA> = X_UserAccess
- "Y" = "Yes"
- "N" = "No"

You can mix compact format and full format syntaxes in the same profile.

Profile Compression

To further reduce the size, you can compress profiles with gzip before sending to the phones. If you encrypt the profiles, you must apply encryption AFTER gzip compression.

Phone Parameters for Remote Provisioning

A number of parameters control how the phone should pull a profile from the provisioning server. The following table summarizes these parameters. See the *Polycom VVX Business IP Phones, OBi Edition Administrator Guide* for a complete reference of available parameters.

Parameters for Remote Provisioning

Parameter Name	Description
X_DeviceManagement.ITSP Provisioning.Method	This parameter controls if and when the phone should download the latest profile from the provisioning server. Choices are: • Disabled = Do not attempt to download profile • System Start = Download from **ConfigURL** just once on system start • Periodically = Download from **ConfigURL** on system start, and then periodically at the interval specified in the Interval parameter Note: The first download on system start is performed after a random delay of 30 to 90 seconds.
X_DeviceManagement.ITSP Provisioning.Interval	When method is set to Periodically, this is the number of seconds between downloads from the provisioned **ConfigURL**. If the value is 0, the phone downloads just one time only on system start (equivalent to setting the method to System Start).
X_DeviceManagement.ITSP Provisioning.ConfigURL	In the simplest form, this can be just a URL to download the profile such as: http://prov-server.myitsp.com/obi$MAC.xml The full syntax is a provisioning script as described in the Provisioning Scripts section.
X_DeviceManagement.ITSP Provisioning.SPRM0 to ***X_DeviceManagement.Provisioning.*SPRM7**	Nonvolatile special parameters that can be used in a provisioning script and can be changed by provisioning only. The **SPRM*n*** values aren't accessible via the web management UI. These can be used in the **ConfigURL** as the option of SYNC Command only. Please see the Provisioning Scripts section for details.

Parameters for Remote Provisioning

Parameter Name	Description
X_DeviceManagement.ITSP Provisioning.GPRM0 to ***X_DeviceManagement.ITSP Provisioning.*GPRM7**	Nonvolatile general parameters that can be used in a provisioning script and can be changed by both the remote provisioning and web management interfaces.
X_DeviceManagement.ITSP Provisioning.TPRM0 to ***X_DeviceManagement.ITSP Provisioning.*TPRM3**	Volatile parameters that can be used in a provisioning script, and can be changed by both remote provisioning and web management interfaces. On system reboot, the **TPRM*n*** parameters are cleared.

Provisioning Scripts

A provisioning script can be used in a **ConfigURL** and **FirmwareURL** parameter. A provisioning script is a sequence of statements separated by a semicolon (;). The phone executes the statements sequentially. The format of a statement is:

```
*<SPNL> [@label 1*<SP>] [IF 1*<SP> ( 1*<SP> expr 1*<SP> ) 1*<SP>] oper [1*<SP>
args] ;
```

Where:

Provisioning Script Parameters

Parameter Name	Description
[…]	An optional element in the syntax.
<SP>	A white space, which can be a space (0x20) or a tab (0x09).
*<SP>	Zero or more <SP>.
1*<SP>	One or more <SP>.
<SPNL>	A <SP> or a newline (0x0A or 0x0D) character.
*<SPNL>	Zero or more <SPNL>.
@label	@ (0x40) followed by a string made up of ASCII characters in the set [a-zA-Z0-9]. A label is used with a GOTO operation: GOTO label.
IF	The string IF (0x49 0x46), which must be followed by (expr). This tells the phone to execute the following operation in the statement only if the condition specified in expr is matched.

Provisioning Script Parameters

Parameter Name	Description
expr	An expression enclosed in parentheses. The format of expr must be $MacroName <SP> ComparisonOperator <SP> Value. where: MacroName is the name of a defined macro, such as TPRM0 or MAC. Value can be a combination of ASCII string and macros, or a quoted string. Examples: • Abcde • DMMAC • abcde${MAC}.xml "abc def higjk" It may not contain any <SP> characters except when enclosed by double quotes. The enclosing double quotes are excluded when doing the comparison. ComparisonOperator is one of the following relational operators: • == (Equal) • != (Not equal) • >= (Greater than or equal to) • > (Greater than) • <= (Less than or equal to) • < (Less than) The definition of the relational operation follows that of the standard C library function strcmp(char *str1, char *str2).
Oper	One of the following operations: SYNC, FWU, WAIT, EXIT, GOTO, SET, CLR. The Provisioning Script Operations section describes these operations.

Notes:

- All statement syntaxes are case-sensitive.
- The maximum size of a script is 2048 bytes. If the size is too big, the script is truncated, execution may be terminated prematurely, and behavior will be unpredictable. Please make sure your script is within this size limit.
- You must also not have any newline character anywhere in a statement other than at the beginning of each statement.

You can use $TPRM0, $TPRM1, $TPRM2, and $TPRM3 as variables to store temporary values in a script. However, be cautious that $TPRM0 may be used by the system to store the result of an operation and may accidentally overwrite the value you explicitly set for it. By default, the SYNC and the FWU operations store the result (1 for success and 0 for failure) in $TPRM0.

Provisioning Script Operations

SYNC

This operation synchronizes the phone's profile with a profile specified by the URL. The phone downloads the specified profile according to the URL, and then decrypts the profile. This operation can be used only in a **ConfigURL** parameter, and must never be used in a **FirmwareURL** parameter.

Syntax:

```
[SYNC 1*<SP>] [-T=var 1*<SP>] [-A=crypto 1*<SP>] [-K=key 1*<SP>] [-IV=iv <SP>] URL
```

Where:

Provisioning Script Parameters

Parameter	Description
`var`	A TPRM*x* to store the result, where *x* = 1, 2, or 3. By default, the result is stored in TPRM0.
`crypto`	`aes` OR `rc4` (the `crypto` to decrypt the profile). Specify `aes` for AES128 or `rc4` for RC4-128.
`key`	The decryption key specified as a 32-character (case insensitive) hex string, such as: `000102030405060708090a0b0c0d0e0f` In case of AES128, `key` should be 128 bits (or 16 bytes or 32 hex digits) long. It is permissible to specify a shorter key. In this case, the phone pads the key with zeros to form a 128-bit key. For RC4, the given key MUST be exactly 128 bits long.
`iv`	The IV for AES128 CBC, specified as a 32-character (case insensitive) hex string, such as `000102030405060708090a0b0c0d0e0f0` `iv` is not needed for RC4. It is optional for AES. If not specified, the phone uses an all-zero string as the IV.
`URL`	The URL to download the profile. HTTP, HTTPS, and TFTP schemes are supported.

Note that in the context of a **ConfigURL** parameter, the opcode SYNC is implied if omitted.

Result:

- 0 (for Failure)
- 1 (for Success)

The operation returns 0 to indicate a failure if one of the following occurs:

- An invalid URL is specified.
- Hostname in the URL cannot be resolved.
- Timeout while waiting for a response from the server. In case of TFTP, the phone retransmits a request every second until a response is received. If no response is received after 30 retransmissions, it is considered a timeout. In case of HTTP and HTTPS, the server must accept the connection request from the phone within 60 seconds and the profile download must be completed within 600 seconds. Otherwise, it is considered a timeout.

- An error code is returned by the server. In case of TFTP, all non-zero error codes are considered errors. In case of HTTP and HTTPS, all HTTP failure response codes are considered errors except 302 and 307 for redirection. The phone honors the redirection response (302 or 307) as many as five times. Beyond that it also is considered an error.
- In case of HTTPS, the server's SSL certificate is invalid (expired or failed verification).
- Profile has invalid format, such as malformed XML or <ParameterList> element not found.

Otherwise, the operation returns 1 to indicate a success. This includes the case where the profile doesn't update any parameters because the profile is empty or the parameters all have the same values as what are currently stored on the phone.

Examples:

```
SYNC –T=TPRM1 –A=aes –K=$SPRM0 –IV=$SPRM1 http://server.mycompany.com/profile.xml
SYNC -A=rc4 -K=$SPRM1 http://192.168.15.102/2003C5-e.cfg
```

RPT (Report Configuration and Status)

This operation lets the phone report configuration and status to a cloud server.

The Sub Command (RPT) in ITSP ConfigURL script is used to exercise this feature.

Syntax:

```
[RPT 1*<SP>] [-stid=var 1*<SP>] [-A=aes 1*<SP>] [-K=key 1*<SP>] [-IV=iv <SP>] [-z=comp]
URL
```

Where:

Script Parameters

Parameter	Description
`var`	Type of configuration to report (*config* or *status*), where *config* = Entire configuration including status (same as the backup file retrieved from webpage without status and default values) *status* = The run status only (same as the "System Status" page retrieved from device webpage)
`crypto`	`aes` only (128-bit CBC)
`key`	The decryption key specified as a 32-character (case insensitive) hex string, such as: `000102030405060708090a0b0c0d0e0f` For AES128, `key` must be 128 bits (or 16 bytes or 32 hex digits) long. It is permissible to specify a shorter key. In this case, the phone pads the key with zeros to form a 128-bit key.
`iv`	The IV for AES128 CBC, specified as a 32-character (case insensitive) hex string, such as `000102030405060708090a0b0c0d0e0f0` `iv` is not needed for RC4. It is optional for AES. If not specified, the phone uses an all-zero string as the IV.

Script Parameters

Parameter	Description
URL	The URL to upload the profile. HTTP, HTTPS, and TFTP schemes are supported.
comp	Compression method (1 or 2). When omitted, no compression is performed on data, Otherwise, use 1 for gzip file or 2 for gz compressed string. Note: When the compression and encryption are both selected, the configuration is compressed first before the encryption is applied. The server needs to process the data accordingly.

In conjunction with a SYNC command, the provisioning server would be able to update profile for device to download according to the current configuration reported by the device. The following ConfigURL example below tells the device to report the current configuration first, then wait for 30 seconds before downloading the profile.

```
RPT -stid=config -z=1 http://192.168.1.28/dev-report.php?obi=$OBN;
WAIT 30;
SYNC http://192.168.1.28/dev-profile.php?obi=$OBN
```

FWU (Firmware Update)

This operation lets the phone update the firmware to the one specified in the given URL. This operation can only be used in a **FirmwareURL** parameter and must not be used in a **ConfigURL** parameter.

Syntax:

```
[FWU 1*<SP>] [-T=var 1*<SP>] URL
```

Where:

- var = A TPRMx to store the result, where *x* = 1, 2, or 3. By default, the result is stored in TPRM0
- URL = URL to download the firmware. HTTP and TFTP schemes are supported

Note that in the context of the **FirmwareURL** parameter, the opcode FWU is implied if omitted. Example:

```
IF ( $FWV <= 1.0.3.1626 ) FWU
http://server.mycompany.com/VVX250-1-0-3-2010-12-5.fw
```

In this example, the phone is updated to the firmware at the given URL only if the current firmware version is older than 1.0.3.1626

Result:

- 0 (for Failure)
- 1 (for Success)

The operation returns 0 to indicate failure if one of the following occurs:

- An invalid URL is specified.
- Hostname in the URL cannot be resolved.
- Timeout while waiting for a response from the server. In case of TFTP, the phone retransmits its request every second until a response is received. If no response is received after 30 retransmissions, it is considered a timeout. In case of HTTP and HTTPS, the server must accept the connection request from the phone within 60 seconds and the profile download must be completed within 600 seconds. Otherwise, it is considered a timeout.

- An error code is returned by the server. In case of TFTP, all non-zero error codes are considered an error. In case of HTTP and HTTPS, all HTTP failure response codes are considered errors except 302 and 307 for redirection. The phone honors the redirection response (302 or 307) as many as five times. After that, it also is considered an error.
- In case of HTTPS, the server's SSL certificate is invalid (expired or failed verification).
- Firmware file has invalid format.
- Firmware file doesn't pass checksum validation. The file may be corrupted.

WAIT

Suspend the execution of the script for at least the specified duration seconds. During this time, the script engine is considered IDLE, which means that a graceful reboot of the system can take place while the script execution is suspended. This is the point where the script stops and lets other scripts start or resume.

Syntax:

```
WAIT 1*<SP> duration
```

Where:

`duration` = the number of seconds to wait before resuming execution.

Example:

```
WAIT 60
```

Wait for 60 seconds before executing the next statement in the script.

EXIT

Stop the execution of the current script.

Syntax:

```
EXIT
```

GOTO

Change the sequence of script execution by jumping to the statement marked with the given @label.

Syntax:

```
GOTO 1*<SP> label
```

Example:

```
@retry IF(xxx) -T=var http://myserver.mycompany.com/obi${MAC}.xml;
 IF ( $TPRM0 == 1 ) EXIT;
WAIT 60;
GOTO retry
```

In the example, the phone synchronizes with the profile at the given URL. Note that we also use the default result variable TPRM0. If the profile downloads successfully when this script executes, it exits and stops executing the task. Otherwise, it waits for 60 seconds and tries again.

SET

Set a variable to the given value

Syntax:

```
SET 1*<SP> TPRMx 1*<SP> = 1*<SP> value
```

Where:

`x` = 0, 1, 2, or 3

`value` = a combination of ASCII strings and macros; it must not contain any `<SP>` characters.

Example:

```
SET TPRM1 = ABC
SET TPRM3 = abcde${TPRM1}
```

CLR

Clear a variable.

Syntax:

```
CLR 1*<SP> TPRMx
```

Where:

`x` = 0, 1, 2, or 3

Operation Error Codes

You can use the $ERR macro in a provisioning script to get the 3-digit error code for the last SYNC or FWU operation. The following error codes are defined:

- Any HTTP failure response codes returned by the server if using HTTP or HTTPS, such as 403, 404, 503
- 801: Transmission time out
- 802: Connection time out
- 803: SSL connection time out
- 805: Too slow (can not get complete file with 10min)
- 806: Server rejects connection
- 810: Server close connection while transmission
- 815: Cannot follow http redirect (bad URL)
- 816: Being redirected more than 4 times
- 820: SSL server name doest not match
- 830: Buffer overflow (internal)
- 831: Out of memory (internal)
- 850: Invalid URL
- 851: Cannot resolve the server name as specified in the URL
- 861: Firmware checksum error

- 862: Firmware downgrade to the specified version is prohibited in the current running version
- 863: Profile is malformatted
- When using TFTP, the following codes may be reported:
  - 500: TFTP code 0 (Unknown error)
  - 404: TFTP code 1 (File not found)
  - 401: TFTP code 2 (Access violation)
  - 405: TFTP code 4 (Bad operation)
  - 400: TFTP code 5 (Unknown TID)
  - 200: The last operation is successful

Provisioning Script Examples

Example 1: (FirmwareURL) Upgrade to a Specific Firmware

```
tftp://server.myinc.com/VVX250-1-0-2-1512.fw;
```

Note that opcode FWU is implied in this simple case.

Example 2: (ConfigURL) Sync to a Specific Profile

```
http://server.myinc.com/$DM-generic.cfg;
```

Note that opcode SYNC is implied in this simple case.

Example 3: (FirmwareURL) Upgrade to Specific Firmware Based on Current Version

The phone updates to firmware version 1.0.2.1512 if its current version is older than that. Otherwise it updates to firmware version 1.0.3.1719 if its current version is older than that. In case the upgrade fails, the phone retries in 60 seconds. Note that the phone reboots if it updates to version 1.0.2.1512. On bootup, it runs the same script again and updates to 1.0.3.1719.

```
@start SET TPRM2 = 2;
IF ( $FWV < 1.0.2.1512) FWU -T=TPRM2 tftp://server.myinc.com/VVX250-1-0-2-1512.fw;
IF ( $TPRM2 == 1 ) EXIT;
IF ( $TPRM2 == 0 ) GOTO error;
IF ( $FWV < 1.0.3.1719 ) FWU -T=TPRM2
tftp://server.myinc.com/VVX250-1-0-3-1719.fw; IF ( $TPRM2 != 0) EXIT;
@error WAIT 60;
GOTO start;
```

Example 4: (ConfigURL) Download with Two Profiles Sequentially

The phone downloads the two given profiles in succession. The changes apply only when the entire script is completed.

```
SYNC http://server.myinc.com/$DM-generic.cfg; SYNC
http://server.myinc.com/$DSN.cfg
```

Example 5: (ConfigURL) Retry Sync with Exponential Back-Off

The phone attempts to download the given profile as many as four times until successful. It waits twice as long as before on each retry, starting with 30 seconds. When it fails after four tries, it waits for an hour before retrying from the beginning again.

```
SET TPRM1 = 0;
@start SYNC http://server.myinc.com/$DM-generic.cfg;
IF ( $TPRM0 == 1 ) EXIT;

IF ( $TPRM1 == 3 ) SET TPRM1 = 4;
IF ( $TPRM1 == 2 ) SET TPRM1 = 3;
IF ( $TPRM1 == 1 ) SET TPRM1 = 2;
IF ( $TPRM1 == 0 ) SET TPRM1 = 1;
IF ( $TPRM1 == 1 ) WAIT 30;
IF ( $TPRM1 == 2 ) WAIT 60;
IF ( $TPRM1 == 3 ) WAIT 120;
IF ( $TPRM1 == 4 ) SET TPRM1 = 0;
IF ( $TPRM1 == 4 ) WAIT 3600;
GOTO start;
```

Script Execution Model

Each provisioning script stored in the phone (**ConfigURL** and **FirmwareURL**) has its own execution thread with an internal execution state. The execution state can be either:

- **Idle**: The script isn't running at the moment and isn't about to start.
- **Ready**: The script can start or resume as soon as no other threads are running.
- **Running**: The script execution is active.
- **Suspended**: The script execution is suspended (inside a WAIT operation).

When a script is about to start, its thread goes from the Idle state to the Ready state. Once the system has determined that the Ready thread can run, it transitions to the Running state. Then it goes from Running to Suspended state when it hits a WAIT operation, or back to Idle when it hits an EXIT operation or the end of script. It can go from Suspended to Ready state when the WAIT timer expires.

A script can be configured to run just once at boot up, or in addition, to run periodically afterwards at regular intervals (such as once every hour). When it's time for the thread to run, the execution state goes from Idle to Ready. When the system boots up, the system executes a WAIT operation on behalf of each script with a nonzero random delay. Therefore all scripts are in the Suspended state when the system starts. The random delay is in the range of 0 to 30 seconds for the **FirmwareURL** script and in the range of 30 to 90 seconds for the **ConfigURL** script. In other words, the **FirmwareURL** script is guaranteed to run first.

By design no more than one script execution thread can assume the Running state at any time. When the current Running thread goes to Idle or Suspended state, the system picks one of Ready threads to run. If there are more than one Ready threads, the **FirmwareURL** script has priority over the **ConfigURL** script.

The phone's provisioning engine is considered busy any time when there is at least one script execution thread is Running. Otherwise, it's considered idle. If the provisioning engine is busy, a request to gracefully reboot the system (for any reason) is postponed until the engine becomes idle again.

Note that there can be two **ConfigURL** scripts defined in the phone, one for ITSP provisioning and one for OBiTALK provisioning. The ITSP provisioning **ConfigURL** script has higher priority over the OBiTALK provisioning **ConfigURL** script.

Phone Behavior on Processing a Profile

As soon as the phone downloads a profile as a result of executing an explicit or implicit SYNC operation in the **ConfigURL** script, it processes the file as follows:

- Decrypt the file according to the SYNC command options, if necessary. Otherwise, check if the file is encrypted by the OBi default encryption and decrypt it accordingly.
- Check if the file is compressed, and run gunzip on it accordingly.
- Parse the XML syntax and discard the profile if it isn't well formed.
- Check if the root element is `<ParameterList>` or else discard the file.
- Check If the `<ParameterList>` element has an **X_Reset** attribute and apply it accordingly (but no reboot yet at this time).
- Parse each `<Object>` element inside `<ParameterList>`. Ignore objects with unrecognized names.
- Parse each `<ParameterValueStruct>` element inside each known object. Ignore parameters with unrecognized names or invalid values. Save the parameter values that are valid and different from the currently stored values.
- All unrecognized XML elements and attributes in the profile are ignored.

Note that the phone doesn't automatically retry a SYNC (or FWU) operation if the operation has failed. It has to be told explicitly in the script to perform a new SYNC (or FWU) following a failure, perhaps after an optional WAIT operation. See the Provisioning Scripts section for an example of retrying SYNC with exponential back-off.

When the script reaches the end or hits a WAIT or EXIT operation, the phone gracefully reboots itself if **X_Reset** has been seen at least once, or if one or more parameters updated, unless the updated parameters so far are all from the following list:

- **X_DeviceManagement.Syslog.Server**
- **X_DeviceManagement.Syslog.Port**
- **VoiceService.1.VoiceProfile.1.Line.1.X_SipDebugOption**
- **VoiceService.1.VoiceProfile.1.Line.1.X_SipDebugExclusion**
- **VoiceService.1.VoiceProfile.1.Line.2.X_SipDebugOption**
- **VoiceService.1.VoiceProfile.1.Line.2.X_SipDebugExclusion**
- **VoiceService.1.VoiceProfile.1.Line.3.X_SipDebugOption**
- **VoiceService.1.VoiceProfile.1.Line.3.X_SipDebugExclusion**
- **VoiceService.1.VoiceProfile.1.Line.4.X_SipDebugOption**
- **VoiceService.1.VoiceProfile.1.Line.4.X_SipDebugExclusion**
- **VoiceService.1.VoiceProfile.1.Line.5.X_SipDebugOption**
- **VoiceService.1.VoiceProfile.1.Line.5.X_SipDebugExclusion**
- **VoiceService.1.VoiceProfile.1.Line.6.X_SipDebugOption**

- **VoiceService.1.VoiceProfile.1.Line.6.X_SipDebugExclusion**

A graceful reboot is one that waits until the system becomes idle (no active calls and provisioning engine idle) before rebooting. The above list of parameters can take effect without a reboot after provisioning. In other words, you can remotely turn on debug on the phone without causing it to reboot also. This would be very useful if you are debugging an active call.

A complete system reboot takes from 30 to 60 seconds to complete. A voice-only reboot is usually sufficient for most parameter changes. A complete system reboot is performed if one or more of the following parameters are changed:

- LAN **OperationMode**
- Any of the network settings
- The **X_Reset** attribute is included in the `<ParameterList>` element of the configuration file

Force Device Sync with SIP NOTIFY

As mentioned earlier, remote provisioning relies on the phone to initiate downloading of the profile. A simple mechanism for the service provider to force the phone to sync up the configuration immediately is to force it to reboot. You can do this remotely by sending down a SIP NOTIFY request to the phone with Event header set to `Reboot`. The reboot is a graceful one. That is, the phone waits until there are no more calls and it is on-hook before it proceeds to reboot.

The `Event:Resync` can be used instead of `Event:Reboot`. In this case, the phone just downloads the profile according to the current **ConfigURL** without rebooting first.

The SIP NOTIFY mechanism may present a security threat and the feature may be disabled completely in the configuration profile. You can mitigate this threat by placing the phone behind a firewall, or by enabling the phone to challenge the request with the same user-id and password provisioned on that user account.

Firewall Considerations

Most phones sit behind a firewall. Normally it is not possible to send down an unsolicited SIP NOTIFY request to the phone, unless a pinhole has been punched through the firewall by the phone to allow the request to get in. Such pinhole is available if the phone is currently registered with the service provider. Registration is done periodically by the phone with an interval specified by the service provider. The interval can be set small enough so that the pinhole remains open to the service provider between registration renewals. Note that the pinhole is only available to the server where the registration is sent. In other words, only the same registration server can send a SIP NOTIFY to the phone through the same pinhole to cause the phone to reboot.

Creating Profiles for Deployment

There are obviously many ways to create a device profile. The choice largely depends on your work flow and the available tools with which you are comfortable. Here, we suggest a few methods to help you get started. Once you become more familiar with the technology, you can develop your own tools to further optimize and streamline the profile creation process.

As each phone can be used by a different end user with different credentials, the final deployment profile for each phone would be different. However, most of the configuration parameters in the profile would still be

the same for all phones. One strategy is to create a profile template with all the generic parameters, and then substitute just a few of the parameters with individualized settings, such as **AuthUserName** and **AuthPassword**, to produce the final profile for each phone.

It is not necessary to include all parameters in the profile. To reduce the size of a profile, you can include only the parameters that you need for your deployment. You can either set the rest of the parameters to default values once when you provision the phone for the first time, and subsequently include a small subnet of parameters in the day-to-day profile. You can use the **X_Reset** attribute in the <ParameterList> (root) element in a profile to force the phone to do a one-time factory reset of all parameters (refer to the Device Profile Format section).

However, you MUST NOT include this **X_Reset** syntax in the day-to-day profile, since that resets all the user settings and causes a complete system reboot.

If you have any questions, ask for assistance from Obi.SPSupport@Polycom.com. Polycom support is here to help to make your deployment a success.

Backing Up a Profile from the Phone Web Page

The phone's current configuration can be backed up and stored as a file in XML format at a user-specified location. The default name of the file is "backup<mac>.xml", where <mac> represents the 12-digit MAC address of unit. When backing up a phone's configuration, you can select the following three options before clicking the **Backup** button.

Backup Options

Option	Description
`Incl. Running Status`	If checked, the values of all status parameters are included in backup file. Otherwise, status parameters are excluded from the backup
`Incl. Default Value`	If checked, the default values of parameters are included in the backup file. Otherwise, default values are excluded from the backup
`Use OBi Version`	If not checked, the backup file uses XML tags that are compliant with TR-104 standard. Otherwise, the backup file is stored in an OBi proprietary format where the XML tags are not compliant with TR-104; but the file size is smaller and the file is more readable

Note: All passwords and PINs (that is, all values that are masked on the phone web page) are excluded from the backup file.

Before running the backup, you may configure the parameters required for your deployment on the phone web page, such as **DigitMaps**, **InboundCallRoutes**, **OutboundCallRoutes**, **ProxyServer**, and so forth.

To back up a base profile suitable for provisioning, all three options in the above table should be unchecked.

Use the ITSP Portal on https://www1.OBiTALK.com.

As a service provider customer, you can request a service provider portal account on OBiTALK.com where you can add one or more administrators for your deployment. Your administrator must already have a user account on OBiTALK.com before being added as an administrator.

When an administrator logs in, OBiTALK.com presents a list of phones being managed by the service provider. From there, the administrator can also add more phones to the service provider account to be managed. The administrator can click the **OBiNo** of any of the managed phones to view and change its configuration.

To generate a profile for a particular phone model, add at least one phone of that model to the service provider account. Then click the **OBiNo** of that phone on the **ITSPportal** to reach the **Manage Device** page for that phone. Then make changes to the phone parameters by clicking the **Goto Device Configuration** button. The page layout is similar to the local phone web page. When you complete the configuration, you can return to the **Manage Device** page and click the **Download Device Profile** button to save the profile on your computer. The profile thus generated is very similar to the one backed up from the phone web page, except this one is complete and doesn't omit any passwords or PIN codes.

Create the Profile Manually

As a starting point, you can back up the configuration of your phone (see last section) to create a configuration file. Then you can cut and paste the parameters you want to configure and add your own settings.

Note that unlike entering values on the local phone web page or the ITSP portal's phone configuration pages, you must properly escape all the XML reserved characters when entering values directly into the profile.

Secure Provisioning

For information on requesting a certificate for your phones, certificate management, please go to http://pki.polycom.com and follow the instructions.

For more information on certificate management, please refer to the documentation below:

https://support.polycom.com/content/dam/polycom-support/products/voice/polycom-uc/other-documents/en/device-certificates-on-phones-tb37148.pdf

Also, see pki.polycom.com/pki

Using HTTPS

The most secure way available to download a phone profile from the provisioning server is by using HTTPS. With HTTPS, both the phone and provisioning server can verify the identity of each other. The data exchanged between the phone and the server are encrypted as well, with the encryption keys secretly negotiated between the two parties. The requirement for using HTTPS for provisioning is for the phone and the server to have a properly signed SSL certificate installed.

Device Authentication

Device authentication is optional in HTTPS, but is highly recommended. During HTTPS handshake, the server can verify the device certificate to make sure the phone is authentic: that the phone is genuinely manufactured by Polycom with a unique MAC address assigned by Polycom, among other information. The device certificate is signed by Polycom and installed in the factory. The server must add the Polycom CA in

its verification chain to verify the device certificate. You may request a copy of the Polycom CA certificate by emailing to cert-admin@obihai.com and cc'ing your Polycom sales representative.

Server Authentication

During HTTPS handshake, the phone verifies the provisioning server's certificate to make sure it's authentic: that the server is truly what it claims it is. To do this, the phone must have the CA certificate that signs the server's certificate in its verification chain. Currently the phones support the following CA:

- Subject: CN=Polycom Root CA
- Subject: C=US, ST=California, O=Obihai Technology Inc., CN=Obihai Certification Authority/emailAddress=support@obihai.com
- Subject: C=US, ST=California, O=Obihai Technology Inc., OU=Obihai Trust Network, CN=Obihai Certification Authority/emailAddress=support@obihai.com
- Subject: L=ValiCert Validation Network, O=ValiCert, Inc., OU=ValiCert Class 2 Policy Validation Authority, CN=http://www.valicert.com//emailAddress=info@valicert.com
- Subject: C=US, O=Equifax, OU=Equifax Secure Certificate Authority
- Subject: C=US, O=VeriSign, Inc., OU=Class 3 Public Primary Certification Authority
- Subject: C=SE, O=AddTrust AB, OU=AddTrust External TTP Network, CN=AddTrust External CA Root
- Subject: C=US, ST=UT, L=Salt Lake City, O=The USERTRUST Network, OU=http://www.usertrust.com, CN=UTN-USERFirst-Hardware
- Subject: C=US, O=The Go Daddy Group, Inc., OU=Go Daddy Class 2 Certification Authority
- Subject: C=US, O=DigiCert Inc, OU=www.digicert.com, CN=DigiCert High Assurance EV Root CA
- Subject: C=US, O=GTE Corporation, OU=GTE CyberTrust Solutions, Inc., CN=GTE CyberTrust Global Root
- Subject: C=US, O=Starfield Technologies, Inc., OU=Starfield Class 2 Certification Authority
- Subject: C=IE, O=Baltimore, OU=CyberTrust, CN=Baltimore CyberTrust Root
- Subject: C=US, O=GeoTrust Inc., CN=RapidSSL SHA256 CA - G3
- Subject: C=US, ST=Arizona, L=Scottsdale, O=GoDaddy.com, Inc., CN=Go Daddy Root Certificate Authority - G2
- Subject: C=US, O=GeoTrust Inc., CN=GeoTrust Global CA
- Subject: C=US, O=VeriSign, Inc., OU=VeriSign Trust Network, OU=(c) 2006 VeriSign, Inc. - For authorized use only, CN=VeriSign Class 3 Public Primary Certification Authority - G5
- Subject: C=US, O=thawte, Inc., OU=Certification Services Division, OU=(c) 2006 thawte, Inc. - For authorized use only, CN=thawte Primary Root CA
- Subject: C=US, O=thawte, Inc., OU=Certification Services Division, OU=(c) 2008 thawte, Inc. - For authorized use only, CN=thawte Primary Root CA - G3
- Subject: C=US, O=DigiCert Inc, OU=www.digicert.com, CN=DigiCert Global Root CA
- Subject: O=Digital Signature Trust Co., CN=DST Root CA X3
- Subject: OU=GlobalSign Root CA - R2, O=GlobalSign, CN=GlobalSign

Server authentication is required and cannot be disabled if HTTPS is used. The service provider must make sure that their provisioning server has a certificate that is signed by one of the above CA. Polycom can also

sign a server certificate for you upon request. The next section describes the steps you can take to prepare a certificate to be signed by Polycom.

Requesting an SSL Certificate from Polycom

You generate a certificate signing request directly from the Polycom device.

By default, the phone requests a 2048-bit certificate with 'sha256WithRSAEncryption' as the signature algorithm. You can use OpenSSL or another certificate signing request utility if you require a stronger certificate.

Polycom supports the use of Subject Alternative Names (SAN) with TLS security certificates. Polycom does not support the use of the asterisk (*) or wildcard characters in the Common Name field of a Certificate Authority's public certificate. If you want to enter multiple hostnames or IP addresses on the same certificate, use the SAN field.

You must have a provisioning server in place before generating the certificate signing request.

For more information, see:

https://documents.polycom.com/bundle/ucs-ag-5-8/page/c_ucs_ag_certificates.html

Using Encrypted Profiles

HTTPS might incur heavy CPU load on the server. A more scalable design is to use HTTP or TFTP but with the configuration files pre-encrypted with a shared secret key. The secret key must be preconfigured on the phone.

The phones support two cryptos for profile encryption: AES128 (CDC with PCKS#5 padding) and RC4. When using a pre-encrypted configuration file, you can specify the crypto, the secret key, and IV as arguments of a SYNC operation in the **ConfigURL** parameter. See the Provisioning Script Operations section for details).

To encrypt the profile, you can use openssl or a similar tool. For example, with openssl, use the following command line for AES encryption:

```
$ openssl enc -aes-128-cbc -K 000102030405060708090a0b0c0d0e0f
     -iv 000102030405060708090a0b0c0d0e0f0 -in plaintext.xml -out encrypted.cfg
```

Use this command line for RC4 encryption:

```
$ openssl enc -rc4 -K 000102030405060708090a0b0c0d0e0f
     -iv 0 -in plaintext.xml -out encrypted.cfg
```

In the last example, IV is not required for RC4 encryption. Still, it must be provided in the command line, but the value can be set to anything, such as 0, as shown.

Instead of using AES or RC4 with a preconfigured shared secret key, you can encrypt the profile without first passing a predefined secret key down to the phone. This is the OBi Default Encryption method, where the encryption algorithm is proprietary and the secret key is derived by each phone internally based on its MAC address. To encrypt a profile using this method, you must use the obicrypt command-line tool available for free from Polycom. Currently, obicrypt is available on Linux and Windows platforms. The command-line syntax follows:

```
     $ obicrypt -M=<mac> [-O=<filename>] <profile>
```

Where:

- <mac> = the 12-digit MAC address (case insensitive) such as 009a1234fAbC.
- <filename> = file name of the encrypted profile (optional).
- <profile> = file name of the plain text profile.

If <filename> is not specified, the encrypted output is stored in the file: obi<mac>.cfg where <mac> is the MAC address. Note that if there is already a file with the same name as the output file name, the tool overwrites the existing file without any warning. Note also that the same tool cannot be used to decrypt the encrypted profile. The only way to verify the contents of the encrypted profile is by loading the profile into the phone with the same MAC address and checking the contents from the phone web page.

It should be advised that the phone's default encryption is NOT as secure as the AES/RC4 method with a shared secret key. It is nevertheless a good method for one-time provisioning of the phone with a shared key to prepare it for subsequent standard AES/RC4 encryption. However, the more secure way is to set the secret key on the phone by initially provisioning it once with HTTPS. It is recommended to store the secret key in one of the SPRM*x* parameters, and reference it in the **ConfigURL** with its corresponding $SPRM*x* macro.

Automating Phone Preparation for Deployment

Without customization, the service provider may need to perform some basic configuration before shipping out units to end users. The service provider may take advantage of these default values for provisioning parameters to facilitate this process:

```
X_DeviceManagement.ITSPProvisioning.Method = System Start
X_DeviceManagement.ITSPProvisioning.ConfigURL = tftp://$DHCPOPT66/$DM.xml
```

Hence by default, the phone attempts to download a generic profile once when the system boots up. The macro $DHCPOPT66 is expanded into option 66 offered by the (local) DHCP server. If DHCP is disabled on the phone or the server does not offer the option, this value is undefined. If the value is defined and is a valid IP address or hostname, the phone executes the **ConfigURL** and downloads the $DM.xml profile. The macro $DM is expanded into the model name of the phone. Note that you need to have a TFTP server listening at the standard port 69 at the option 66 host address to serve the $DM.xml file.

You can put any appropriate information in the generic profile. Typically you would daisy chain multiple profiles such that the final user-specific profile is loaded onto the phone at the last step. The last profile is also the day-to-day profile that the phone regularly grabs from the field. Below we present a simple example to illustrate the rationale for this approach.

As the first profile in the chain, $DM.xml may contain a few parameters to establish some basic boundaries for the phone to operate within. Most importantly, it contains a **ConfigURL** that points to your provisioning server so that you can control the unit after it ships. For example:

```
ConfigURL = https://prov-server.myitsp.com/$MAC-init.xml
```

It's also a good idea to use this opportunity to factory reset the rest of the parameters just to be sure that every parameter is what you expect them to be. For this purpose, you would include the following line in $DM.xml:

```
<ParameterList X_Reset="All">
```

Note that the **X_Reset** parameter causes a complete system reboot and should be used just once in the initialization profile. You should remove it in subsequent profiles to avoid unexpected reboot, and you must never use it in the final day-to-day profile.

You may also want to make sure that the phone is running the firmware version of your choice, say nothing older than version 1.1.0.1891. You can do this by inserting a proper **FirmwareURL** in `$DM.xml`. A listing of `$DM.xml` is shown in the next section.

In practice, the phone can be repackaged and shipped out to the end user once you have verified that it has successfully received `$DM.xml` (by doing the equivalent of opening the phone web page and checking the **ConfigURL**, for instance). You could also wait until the last profile is loaded onto the phone before shipping it out, perhaps to allow your staff to verify everything regarding the user account is in the right order by making a test call.

As shown in the last URL, the second profile in the chain is `$MAC-init.xml` (where `$MAC` is replaced by the actual MAC address in the name of the configuration file for the phone). You should use HTTPS to receive this profile, especially if this step is done outside of your premises or over the public Internet.

The main purpose of `$MAC-init.xml` is to store a secret decryption key in the phone and to let the phone subsequently switch to use the encrypted profile. As shown in the listing `$MAC-init.xml` in the next section, the secret key and the IV are stored in the parameters `SPRM0` and `SPRM1`, respectively. The secret key should be individualized for each phone, hence the need to include `$MAC` in the profile name so that the server can tell which phone is making that request. The crypto to use in this case is AES128, as specified in the **ConfigURL**:

```
ConfigURL= SYNC -A=aes -K=$SPRM0 -IV=$SPRM1
http://prov-server.myitsp.com/$MAC-encrypted.cfg
```

The last profile in the chain is `$MAC-encrypted.cfg`, which contains information specific to the user account. This profile must be encrypted with the secret keys established in `$MAC-init.xml`.

Profile Listings for the Last Example

$DM.xml (Replace `$DM` with the phone name, such as VVX250)

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Generic Configuration File (VVX250.xml) -->
<ParameterList X_Reset=”All”>
<Object>
  <Name>X_DeviceManagement.FirmwareUpdate.</Name>
  <ParameterValueStruct>
    <Name>Method</Name>
    <Value>System Start</Value>
  </ParameterValueStruct>
  <ParameterValueStruct>
    <Name>FirmwareURL</Name>
    <Value>
    IF FWV &lt;  FWU http://prov-server.myitsp.com/VVX250-1-1-0-1891.fw
    </Value>
  </ParameterValueStruct>
</Object>
<Object>
  <Name>X_DeviceManagement.ITSPProvisioning.</Name>
  <ParameterValueStruct>
    <Name>Method</Name>
    <Value>Periodically</Value>
  </ParameterValueStruct>
```

```
  <ParameterValueStruct>
    <Name>Interval</Name>
    <Value>3600</Value>
  </ParameterValueStruct>
  <ParameterValueStruct>
    <Name>ConfigURL</Name>
    <Value> SYNC https://prov-server.myitsp.com/$MAC-init.xml </Value>
  </ParameterValueStruct>
</Object>
</ParamterList>
```

$MAC-init.xml (Replace $MAC with the MAC address, such as 9CADEF000000)

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Unit Specific Initial Configuration File (9CADEF000000-init.xml) -->
<ParameterList>
  <Object>
    <Name>X_DeviceManagement.ITSPProvisioning.</Name>
    <ParameterValueStruct>
      <Name X_UserAccess="noAccess">Method</Name>
      <Value>Periodically</Value>
    </ParameterValueStruct>
    <ParameterValueStruct>
      <Name X_UserAccess="noAccess">Interval</Name>
      <Value>3600</Value>
    </ParameterValueStruct>
    <ParameterValueStruct>
      <Name X_UserAccess="noAccess">ConfigURL</Name>
      <Value>
      SYNC -A=aes -K=$SPRM0 -IV=$SPRM1 http://prov-server.myitsp.com/$MAC-encrypted.cfg
      </Value>
    </ParameterValueStruct>
    <ParameterValueStruct>
      <Name X_UserAccess="noAccess">SPRM0</Name>
      <Value>0102030405060708090a0b0c0d0e0f</Value>
    </ParameterValueStruct>
    <ParameterValueStruct>
      <Name X_UserAccess="noAccess">SPRM1</Name>
      <Value>102030405060708090a0b0c0d0e0f0</Value>
    </ParameterValueStruct>
  </Object>
</ParameterList>
```

$MAC-encrypted.cfg (Replace `$MAC` with the MAC address, such as 9CADEF000000)

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Uner Specific Configuration File (9CADEF000000-encrypted.cfg) -->
<ParameterList>
  <Object>
    <Name>DeviceInfo.</Name>
```

```
    <ParameterValueStruct>
      <Name X_UserAccess="noAccess">ProtectFactoryReset</Name>
      <Value>1</Value>
    </ParameterValueStruct>
  </Object>
  <Object>
    <Name>DeviceInfo.Time.</Name>
    <ParameterValueStruct>
      <Name X_UserAccess="noAccess">NTPServer1</Name>
      <Value>pool.ntp.org</Value>
    </ParameterValueStruct>
  </Object>
  <Object>
    <Name>X_DeviceManagement.WebServer.</Name>
    <ParameterValueStruct>
      <Name X_UserAccess="noAccess">AdminPassword</Name>
      <Value>VVX250Admin@myinc</Value>
    </ParameterValueStruct>
  </Object>
  <Object>
    <Name>X_DeviceManagement.FirmwareUpdate.</Name>
    <ParameterValueStruct>
      <Name X_UserAccess="noAccess">Method</Name>
      <Value>Periodically</Value>
    </ParameterValueStruct>
    <ParameterValueStruct>
      <Name X_UserAccess="noAccess">Interval</Name>
      <Value>3600</Value>
    </ParameterValueStruct>
    <ParameterValueStruct>
      <Name X_UserAccess="noAccess">FirmwareURL</Name>
      <Value>
     IF ( $FWV &lt; 1.0.3.1891 ) FWU http://prov-server.myitsp.com/VVX250-1-1-0-1891.fw
      </Value>
    </ParameterValueStruct>
  </Object>
  <Object>
    <Name>X_DeviceManagement.Provisioning.</Name>
    <ParameterValueStruct>
      <Name X_UserAccess="noAccess">Method</Name>
      <Value>Periodically</Value>
    </ParameterValueStruct>
    <ParameterValueStruct>
      <Name X_UserAccess="noAccess">Interval</Name>
      <Value>3600</Value>
    </ParameterValueStruct>
    <ParameterValueStruct>
      <Name X_UserAccess="noAccess">ConfigURL</Name>
      <Value>
      SYNC -A=aes -K=$SPRM0 -IV=$SPRM1 http://prov-server.myitsp.com/$MAC-encrypted.cfg
```

```
      </Value>
    </ParameterValueStruct>
  </Object>
  <Object>
    <Name>VoiceService.1.VoiceProfile.1.Line.1.SIP.</Name>
    <ParameterValueStruct>
      <Name X_UserAccess="noAccess">AuthUserName</Name>
      <Value>14088906000</Value>
    </ParameterValueStruct>
    <ParameterValueStruct>
      <Name X_UserAccess="noAccess">AuthPassword</Name>
      <Value>1408888888password</Value>
    </ParameterValueStruct>
  </Object>
  <Object>
    <Name>VoiceService.1.VoiceProfile.1.Line.1.CallingFeatures.</Name>
    <ParameterValueStruct>
      <Name X_UserAccess="noAccess">CallerIDName</Name>
      <Value>John J. Smith</Value>
    </ParameterValueStruct>
  </Object>
  <Object>
    <Name>VoiceService.1.VoiceProfile.1.</Name>
    <ParameterValueStruct>
      <Name X_UserAccess="noAccess">DTMFMethod</Name>
      <Value>Auto</Value>
    </ParameterValueStruct>
  </Object>
  <Object>
    <Name>VoiceService.1.VoiceProfile.1.SIP.</Name>
    <ParameterValueStruct>
      <Name X_UserAccess="noAccess">ProxyServer</Name>
      <Value>ProxyServer.myinc.com</Value>
    </ParameterValueStruct>
    <ParameterValueStruct>
      <Name X_UserAccess="noAccess">RegistrationPeriod</Name>
      <Value>120</Value>
    </ParameterValueStruct>
  </Object>
</ParameterList>
```