



**DEPLOYMENT GUIDE**

November 2019 | 3725-85484-002

# Poly OBi Voice Adapter Device Deployment Guide

## OBi3 and OBi5 Series

### **GETTING HELP**

For more information about installing, configuring, and administering Poly/Polycom products or services, go to Polycom Support.

Plantronics, Inc. (Poly – formerly Plantronics and Polycom)  
345 Encinal Street  
Santa Cruz, California  
95060

© 2019 Plantronics, Inc. All rights reserved. Poly, the propeller design, and the Poly logo are trademarks of Plantronics, Inc. All other trademarks are the property of their respective owners.

# Contents

<b>Before You Begin</b> .....	<b>5</b>
Audience, Purpose, and Required Skills .....	5
Note to End Users .....	5
Getting Help .....	6
Polycom and Partner Resources .....	6
The Polycom Community .....	6
Documentation Feedback .....	6
<b>Getting Started</b> .....	<b>7</b>
Product Overview .....	7
Canonical Fashion .....	8
Literal Fashion .....	8
Boolean Values .....	9
Multiple Choice Values .....	9
Parameter Values .....	9
<b>Introduction</b> .....	<b>10</b>
Device Parameters and Objects .....	10
Relationship between Objects with Similar Names .....	11
How the Device Organizes SP Account Parameters .....	12
Which Objects to Configure .....	14
Macros .....	15
Parameter Macro Expansion .....	15
User-Defined Macros .....	17
<b>OBi Device Provisioning</b> .....	<b>19</b>
Device Configuration .....	19
Local Device Configuration .....	19
Device Web Page Configuration .....	20
OBi Web Page Banner Customization (OBi 302 only) .....	21
Device IVR Configuration .....	21
Zero-Touch Device Customization .....	22
Factory Reset .....	22
End-User 'User' Parameter Space .....	23
Locking Parameters .....	24

Firmware Update .....	24
Firmware Update Methods .....	24
Update from the Device Web Page .....	24
Update From the IVR .....	24
Update Using the FirmwareURL .....	25
<b>Device Configuration Profile Formats .....</b>	<b>27</b>
Device Profile Format .....	27
Full Profile Format .....	27
Compact Profile Format .....	31
Profile Compression .....	31
Device Parameters for Remote Provisioning .....	31
Provisioning Script .....	32
Provisioning Script Operations .....	34
SYNC .....	34
FWU (Firmware Update) .....	35
WAIT .....	36
EXIT .....	36
GOTO .....	36
SET .....	36
CLR .....	37
Operation Error Codes .....	37
Provisioning Script Examples .....	38
Script Execution Model .....	39
Device Behavior on Processing a Profile .....	40
Force Device Sync with SIP NOTIFY .....	41
Firewall Considerations .....	41
Creating Profiles for Deployment .....	41
Backing Up a Profile from the Device Web Page .....	42
Use the PDMS-SP Cloud Management Portal .....	42
Create the Profile Manually .....	43
Secure Provisioning .....	43
Using HTTPS .....	43
Device Authentication .....	43
Server Authentication .....	44
Requesting an SSL Certificate from Polycom .....	45
Using Encrypted Profiles .....	45
Automating Device Preparation for Deployment .....	46
Example Profile Listings .....	47

# Before You Begin

---

This guide describes how to develop and deploy XML-based applications to configure Polycom® OBi analog telephone adapter (ATA) devices.

The information applies to the following OBi device models:

- OBi3 Series
- OBi5 Series

## Audience, Purpose, and Required Skills

This guide is written for a technical audience involved in developing and deploying XML-based applications for OBi devices. 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
- Open SIP 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 want to deploy and remotely manage OBi devices using a central provisioning system.
- VARs planning to support customers remotely, managing OBi devices via a central provisioning system.
- Power users of OBi devices who want to remotely manage their devices for their friends and family.

This document describes the technologies and methods to manage these devices remotely and to securely provision your devices at a massive scale. A complete listing of available configuration parameters on the device is given at the end of this document. For a complete device parameter reference, please see the [Polycom OBi Device Administrator Guide](#).

## Note to End Users

This guide targets device administrators who manage large installed bases.

End users are highly encouraged to use the OBiTALK web portal at [www.obitalk.com](http://www.obitalk.com) to configure and manage their OBi devices, or to perform management tasks locally by using the device's native web portal,

accessible by logging into the device from any PC with a web browser. See the Polycom® VVX® Business IP Phones, OBi Edition User Guide for more information about your device's native web portal.

## Getting Help

For more information about installing, configuring, and administering Polycom products, see **Documents & Software** at [Polycom Support](#).

## *Polycom and Partner Resources*

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

- Visit the OBi device documents and guides found at <https://documents.polycom.com>
  - *Polycom® OBi Device Administrator Guide*
  - *Polycom® OBi Edition Deployment Guide* (this document)
  - *Polycom VVX Business IP Phones, OBi Edition User Guide*
- For Polycom Software releases and documentation, see [Polycom Voice Support](#).
- For user guides for Polycom voice products, refer to the product support page for your device at [Polycom Voice Support](#).
- For help or technical support for your devices, 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 Service Provider Support Team at: [Obi.SPSupport@Polycom.com](mailto: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.

# Getting Started

---

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

The terms “the device” and “your device” refer to any of the Polycom OBi ATA devices in the OBi3 and OBi5 Series. Unless specifically noted in this guide, all device models operate in similar ways.

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

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

Typical applications include:

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

## Product Overview

Built with a high-performance system-on-a-chip platform to ensure high-quality voice conversations, OBi devices are dedicated systems targeted at applications for Voice over IP services. OBi devices have high availability and reliability because they’re always on to make or receive calls. A computer doesn’t need to be on to talk to people. An OBi device doesn’t require a computer to use. To get started, all you need is a touch tone phone, power, and a connection to the Internet.

This guide describes the following OBi devices:

- OBi3 Series: OBi300/302/312
- OBi5 Series: OBi504/508

**OBi Devices**

OBi Device Model	VoIP Accounts	OBiTALK Support	PHONE Ports	LINE Ports	GigE Ports	Fast E-net Ports	USB Ports	Audio Line In	Audio Line Out	PSTN Failover Ports
OBi300	4	Yes	1	0	0	1	1	0	0	0
OBi302	4 SIP only	Yes	2	0	0	2	1	0	0	0
OBi312	4	Yes	1	1	1	1	1	0	0	0
OBi504	9	Yes	4	0	1*	3*	3	1	1	1**
OBi508	9	Yes	8	0	1*	3*	3	1	1	1**

\* You can configure a OBi5 Series device as a router or as a switch. In router mode, the Gigabit Ethernet port functions as the WAN port, and the three Fast Ethernet ports as three LAN switch ports. In switch mode, all four ports function as switch ports.

\*\* For failover during power outage only.

## 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 include 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

Provisioning or writing OBiPhoneXML apps uses the literal fashion.

- `<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.



You can use more than one `<ParameterGroupName>` which is case-sensitive. The string must not contain any spaces and end with a period.

The `<ParameterName>`. is the name of the parameter, and is always terminated with a period, as shown. This string must not include 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` isn't 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 `<Parameter.Group.Name>.<ParameterGroupName>.<ParameterName>.Parameter Value`. However, text formatting such as bold face isn't required and will be removed when your script or app processes.

OBiPhoneXML apps can also be launched with PDMS-SP (Poly Device Management Services - for Service Providers). This gives the added benefit of using this feature through the cloud without the need for the service provider to host their own server.

## Boolean Values

You can identify parameters that take a Boolean value on your device'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". The only valid Boolean parameter values to use in a device 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

If a parameter has multiple choices, you can either use the default option or provision a value to it.

The options for a parameter are available on the device web page as a drop-down list. Since it's case-sensitive and doesn't allow extra spaces, copy the value from the web page into your provisioning script.

## Parameter Values

When entering a parameter value from the web page or via provisioning, don't add extra 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, don't add extra spaces before and after the delimiter.

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

If a parameter value can include 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`

# Introduction

---

The OBi devices share these same functionalities:

- Support for all standard SIP-based IP PBX and ITSPs/VSPs.
- Suited for all sizes of service providers and enterprise deployment environments.
- Ideal for self-service installations. Home users, small business owners, or corporate IT departments can easily install, set up, and manage these devices.
- Integration with popular softswitch architectures.
- Cloud management enabled via PDMS-SP (<https://www1.obitalk.com>) with both a user portal and an ITSP partner portal with an optional REST API.



The PDMS-SP portal may be used by service providers for device provisioning, management, and troubleshooting. This portal is only accessible by service providers. You can use the PDMS-SP portal independently as the sole system for secure management of OBi devices. You can also use it with an existing centralized provisioning system managed by the service provider.

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

## Device Parameters and Objects

Every OBi device is a highly programmable device with more than a thousand configuration parameters that controls every aspect of its operation. Following the TR-104 standard naming convention, device parameters are grouped into a small number of hierarchical objects. Unique canonical names identify each configuration parameter. They are composed of 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`):

- `VoiceService1.VoiceProfile1.Line1.Enable`

Where `VoiceService1.VoiceProfile1.Line1.` 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's 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 Service - Enable` parameters have the following literal names:

- `VoiceService1.VoiceProfile1.Line2.Enable`

- `VoiceService1.VoiceProfile1.Line3.Enable`
- `VoiceService1.VoiceProfile1.Line4.Enable`

The above shows four instances of the `VoiceService1.VoiceProfile1.Line1` objects in the configuration under the `VoiceService1.VoiceProfile1` 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, and D:

- `VoiceService1.VoiceProfile1.SIP.ProxyServer`
- `VoiceService1.VoiceProfile2.SIP.ProxyServer`
- `VoiceService1.VoiceProfile3.SIP.ProxyServer`
- `VoiceService1.VoiceProfile4.SIP.ProxyServer`

The above shows four instances of the `VoiceService1.VoiceProfile` objects, corresponding to ITSP Profile A, B, C, and D. Note that the `Line` object is only defined under the `VoiceService1.VoiceProfile1` object. It is undefined under the `VoiceService.1.VoiceProfile.2.`, `VoiceService.1.VoiceProfile.3.`, and `VoiceService.1.VoiceProfile.4.` objects. This helps reduce the total number of device 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 aren't described in the TR-104 standard. For these objects and parameters, their names have the prefix `X_` attached to indicate that they're 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`. Don't misinterpret this 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`.

## ***Relationship between Objects with Similar Names***

The use of TR-104 object names is simply to divide the parameter naming space such that the devices can be referenced and managed more conveniently. In general, all objects in the device configuration are typically independent of each other. They don't inherit any properties from their "parent" despite the fact that their names are children of another object in syntax. Sibling objects in this sense also don't share any common properties.

For example, the parameters in the object `VoiceService.1.VoiceProfile.1.Line.1` (parameters under SP1 Service on the device 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 device web page). You can set up an ITSP account on SP1 Service that refers to any of the available ITSP Profiles.

## How the Device Organizes SP Account Parameters

The best way to understand the organization of parameters in your device is to study 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 4. 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. Each SP service contains a parameter that points to an ITSP $x$  profile, where  $x = A$  through D. 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. A device with two user accounts from the same ITSP is configurable on two different SP $n$  services that refer to the same ITSP $x$  profile. Similarly, two SP $n$  services on one device can share the same tone definitions if their parameters point to the same Tone Profile.

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
<i>VoiceService.1.VoiceProfile.1.Line.n.</i> ( $n = 1,2,3,4$ )	Voice Services/SP $n$ Service – SP $n$ Service ( $n = 1,2,3,4$ )	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. ( $n = 1,2,3,4$ )	Voice Services/SP $n$ Service – SIP Credentials ( $n = 1,2,3,4$ )	
VoiceService.1.VoiceProfile.1.Line.n.CallingFeatures. ( $n = 1,2,3,4$ )	Voice Services/SP $n$ Service – Calling Features ( $n = 1,2,3,4$ )	
<i>VoiceService.1.VoiceProfile.n.</i> ( $n = 1,2,3,4$ )	ITSP Profile $x$ /General – General ( $x = A,B,C,D$ corr. $n = 1,2,3,4$ )	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 refers to the four objects as a group. The SP Service parameter <code>X_ServProvProfile</code> binds the SP service to the ITSP profile.
<i>VoiceService.1.VoiceProfile.n.ServiceProviderInfo.</i> ( $n = 1,2,3,4$ )	ITSP Profile $x$ /General – Service Provider Info ( $x = A,B,C,D$ corr. $n = 1,2,3,4$ )	
<i>VoiceService.1.VoiceProfile.n.SIP.</i> ( $n = 1,2,3,4$ )	ITSP Profile $x$ /SIP – SIP ( $x = A,B,C,D$ corr. $n = 1,2,3,4$ )	
VoiceService.1.VoiceProfile.n.RTP. ( $n = 1,2,3,4$ )	ITSP Profile $x$ /RTP – RTP ( $x = A,B,C,D$ corr. $n = 1,2,3,4$ )	
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 <code>X_CodecProfile</code> binds the SP service to the Codec Profile.

### Sample SP Account Parameter Object Mappings

Provisioning Parameter Object	Parameter Group on Device Web Page	Notes
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 <code>X_RingProfile</code> binds the SP service to the Ring Profile.
VoiceService.1.X_FXS.n. (n = 1,2)	Physical Interfaces/PHONE n port (n = 1,2)	A phone port 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 port (or all the phone ports).
VoiceService.1.X_StarCode.n. (n = 1,2)	Star Codes/Star Code Profile x (x = A,B corr. n = 1,2)	A phone port has the <code>StarCodeProfile</code> parameter 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)	A phone port has the <code>ToneProfile</code> parameter to bind to a Tone Profile.
SpeedDial.	User Settings/Speed Dials	Speed Dials are shared among all phone ports, but can be split up among phone ports by creating proper phone digit maps.

An OBi device phone port isn't necessarily bound to just one of the SP Services configured on the device. 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 OBi Device Administrator Guide* for details on using these parameters.

### Parameters Binding a Phone Port to an SP Service

Parameter	Notes
VoiceService.1.VoiceProfile.1.Line.n.InboundCallRoute (n = 1,2,3,4)	<code>InboundCallRoute</code> determines which phone ports to ring on incoming calls on this SP service.
VoiceService.1.X_FXS.n. DigitMap (n = 1,2)	Controls the number patterns users can dial on the phone port.

### Parameters Binding a Phone Port to an SP Service

Parameter	Notes
VoiceService.1.X_FXS.n. OutboundCallRoute (n = 1,2)	Controls which service to make a call based on the dialed number (after validation and transformation with the corresponding phone port digit map).
VoiceService.1.X_FXS.n. PrimaryLine (n = 1,2)	Selects a service (with the default digit map and outbound call route values) which is used to make an outbound call without the need for a user to dial a ** x prefix (where x is 1, 2, 3, 4, 8, or 9).

Note: There is only one `SpeedDial` object per device. For models with two phone ports, the two ports share this object. If you need to restrict each phone port to use a different set of speed dials, you can split up the speed dials into two groups. For example, 1 – 49 can be used from Phone Port 1 only, and 51 – 99 from Phone Port 2 only. By a simple manipulation of the Phone Ports' `DigitMap` parameter, you can still dial 1 – 49 to access the speed dials on either phone port. An example of how this is shown in the following table. Notice how the 1 – 49 numbers are mapped to 51 – 99 respectively on Phone Port 2.

### Phone Port Digit Map for Splitting up the SpeedDial object

DigitMap	DigitMap Rules
VoiceService.1.X_FXS.1. DigitMap	( [1-9]   [1-4][0-9]  ...)
VoiceService.1.X_FXS.2. DigitMap	(<5>[1-9]   <1:6>[1-9]   <2:7>x   <3:8>x   <4:9>x  ...)

Each instance of SP Services, ITSP Profiles, Codec Profiles, Ring Profiles, and Tone Profiles is independent. The instances of the same objects don't 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. Additionally, 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

If you only need to configure one account on the device for the service you offer, select an available SP Service slot (say, SP1) and an available ITSP Profile slot (say, ITSP Profile B). Then, 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 device, 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 device

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 enables you to use macros when provisioning your devices, 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

**Currently Defined Macros**

Macro Name	Value	ExpandIn	Script	Web	Provisioning
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 OBi302	ANY	N	Y	N
DMN	Device Model Number, such as 302	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_DeviceManagem ent.ITSPProvisi oning.ConfigURL and X_DeviceManagem ent.FirmwareUpd ate.FirmwareURL	Y	N	Y
GPRM0 to GPRM7	X_DeviceManagement. ITSPProvisioning.GPRM0 to X_DeviceManagement. ITSPProvisioning.GPRM7	X_DeviceManagem ent.ITSPProvisi oning.ConfigURL and X_DeviceManagem ent.FirmwareUpd ate.FirmwareURL	Y	Y	Y
TPRM0 to TPRM3	X_DeviceManagement.ITSPProvis ioning.TPRM0 to X_DeviceManagement.ITSPProvis ioning.TPRM3	X_DeviceManagem ent.ITSPProvisi oning.ConfigURL and X_DeviceManagem ent.FirmwareUpd ate.FirmwareURL	Y	Y	Y



**Currently Defined Macros**

Macro Name	Value	ExpandIn	Script	Web	Provisioning
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 device 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.myitisp.com/obi\${mac}-signup.xml?code=\${UDM0}



A new subscriber may enter a random code, say 8714, into speed dial 99 by dialing the following star code sequence from a connected device: \*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 device 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 Device Provisioning

---

Your devices are highly programmable devices with more than a thousand configuration parameters on each. Configuration enables you to choose how and when you provision your devices 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.

## Device Configuration

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

- Remote device configuration
- Local device configuration
- Native web page configuration
- Polycom® PDMS-SP Zero-Touch configuration

The parameter set to upload to a deployed device 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 device 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 device is referred to as remote provisioning.

The URL for the device 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 Script](#) section in this document, or refer to the Polycom OBi Device Administrator Guide.

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

## Local Device Configuration

There are two ways to configure the device locally (i.e. without using remote provisioning):

- Browsing the device web pages from a web browser running on a computer.

- Invoking the built-in IVR from a phone attached to a device phone port.

## Device Web Page Configuration

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

- The device is connected to the network with proper IP address assigned.
- You need the current IP address of the device. To do this locally, access the IVR system by entering **\*\*\* 1** from any phone connected to one of the device phone ports. The current IP address is announced.

To log in to the native device web page, browse to the URL `http://Device-IP-Address/`, where `Device-IP-Address` is the current IP address of the device. You can view and change a device's configuration as well as update its firmware on the native web page served locally from the device. This method of device 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 device. 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 device. In fact, most service providers would rather disable this capability on the device so that the end user cannot tamper with its configuration. However, a service provider may still use the device web page in a lab environment when initially experimenting with the device parameter settings for eventual locked-down remote mass-provisioning or to prepare a device before it is shipped out to an end-user, then switch to remote provisioning after deploying the unit.

Access to the device web pages may be protected by passwords. Two passwords can be configured on the device: an Admin Password and a User Password. If a non-empty 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 device serves the pages without prompting for user-id and password.

These four parameters in the **X\_DeviceManagement.WebServer** object control the behavior of the device's built-in web server:

### Parameters That Control the Device's Native Web Server

Parameter	Description
Port	The web server listen port. Default is 80, the standard HTTP port. Setting this value to 0 disables all web server access.
AdminPassword	Admin login password. Default is <code>admin</code> .
UserPassword	User login password. Default is <code>user</code> .
AccessFromWAN	Enables the web server to serve web pages to the WAN side. Default is 0 (disabled). Note: Serving web pages to the LAN side is always allowed and cannot be disabled.

If necessary, you can block end user access to the admin or user device web pages by setting a non-empty 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 device'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 device 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 devices on OBiTALK.com, which enables you to access all your devices without having to enter the URL `http://Device-IP-Address/` for each device. The appearance between the OBiTALK web page and the native device web page is different, but as an admin user you have identical access to settings.

## OBi Web Page Banner Customization (OBi 302 only)

The banner displayed across the top section of the device web page can be customized. The image file for the banner must be in PNG format with a file size no larger than 64 KB. Internally, this image is referred to as `custom-logo.png`. Two hidden parameters (changeable by provisioning only) control the custom banner:

- **X\_DeviceManagement.WebServer.CustomLogoTag** is an HTML fragment that describes how the image should be displayed on the page. For example:

```
<div><a href="http://www.itsp.com" target="_top"> </a></div>
```

Note that the device doesn't check the syntax of this value. Be sure to properly escape the value in accordance with XML standards when entering it directly in a device configuration file.

- **DeviceManagement.WebServer.CustomLogoURL** is a URL that tells the OBi device where and how to download the custom banner image. For example:

```
http://www.itsp.com/image/obi202-logo.png
```

The OBi device attempts to download the image from the given URL at start-up if **CustomLogoTag** is not empty and **CustomLogoURL** is a valid URL that is different from the URL where the currently stored banner image was downloaded from. If the image is successfully downloaded, it is stored in flash memory to replace the last stored one. Otherwise, the device waits 300 seconds before retrying. The stored banner image won't be erased by a factory reset of the device.

## Device IVR Configuration

The prerequisites for accessing the device web pages are:

- The device is connected to the LAN (or WAN) with proper IP address assigned.
- A way to find out the current IP address of the device.

In a typical environment, when the device is physically connected to a network it can be assigned an IP address automatically by a DHCP server. Then you can invoke the device IVR to find out the assigned IP address. Then you can access the IVR from any phone connected to one of the device phone ports by

dialing `***` and selecting one of the options on the main menu. For example, the current IP address is announced by selecting option 1 for Basic Network Status.

There are situations where DHCP is not available and a static IP address must be manually assigned to the device. This can also be done from the IVR using option 4. DHCP also is disabled when a valid IP address is entered and saved under this option.

One can also perform a factory reset of the device from the IVR using option 8 (additional restrictions applied; see the section [Factory Reset](#) for more about this). Other than options 1 (Basic Network Status) and 2 (Advanced Network Status), all other IVR options may be protected with an IVR access password that can only contain digits (0–9), such as 02379.

Note that the number `***` to invoke the IVR is configured in the Phone Port's `DigitMap` and `OutboundCallRoute` parameters. By default each Phone Port has the rule `|***|` in the `DigitMap` and the rule `{***:aa2}` in the `OutboundCallRoute` (where `aa2` is the short name for the internal IVR). By removing these rules one can effectively disable access to the IVR.

## Zero-Touch Device Customization

Polycom offers a device customization service called PDMS-SP Zero-Touch that enables service provider customers to select the default values for device network configuration parameters. Contact [obi.spsupport@polycom.com](mailto: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, enter the IVR (`***/8`), or press the reset button on the back of the device when device is powered on.

You can limit the factory reset parameters with user read-write permission only by setting the `DeviceInfo.ProtectFactoryReset` parameter to 1. Setting it to 0 enables resetting all parameters, regardless the user read-write properties. Note that for this option to work, the `Web Server AdminPassword` parameter must be set to a non-default value (that is, the parameter must not have the “Default” option checked).

Each parameter is categorized as either a `Voice` parameter or a `Router` parameter. The factory reset operation can be further limited to just the voice parameter set or the router parameter set by setting the `DeviceInfo.FactoryResetMode` parameter to `Voice` or `Router`. The default value is `All`, which resets everything.



The `ProtectFactoryReset` and `FactoryResetMode` parameters can only be changed via provisioning. The service provider must take extreme care before setting up these protections, because that limits the reset operation when a user presses the hard-reset button on the unit, and may not recover the unit to factory default condition. The service provider must make sure those non-resettable parameters will not cause harmful effects that hinder the unit from going into normal operation. It is always a good idea to test new settings on some in-house units first before applying them on deployed units.

The following table summarizes the factory reset behavior.

**Parameters That Control Factory Reset Behavior on the Device's Native Web Server**

<b>DeviceInfo. ProtectFactoryReset</b>	<b>DeviceInfo. FactoryResetMode</b>	<b>IVR (***/8) OR Reset Button</b>	<b>IVR (***/0/81#) (Reset Voice Only)</b>	<b>IVR (***/0/82#) (Reset Router Only)</b>
0	All	Reset all parameters	Reset all voice parameters	Reset all router parameters
1	All	Reset all parameters with user read-write enabled	Reset all voice parameters with user read-write enabled	Reset all router parameters with user read-write enabled
0	Voice	Reset all voice parameters	Reset all voice parameters	No effect
1	Voice	Reset all voice parameters with user read-write enabled	Reset all voice parameters with user read-write enabled	No effect
0	Router	Reset all router parameters	No effect	Reset all router parameters
1	Router	Reset all router parameters with user read-write enabled	No effect	Reset all router parameters with user read-write enabled

You also can perform a factory reset from the device web page under **System Management/Device Update**, which allows resetting voice or router parameters only. The `ProtectFactoryReset` and `FactoryResetMode` parameters have no effect when performing factory reset this way. Instead, the current login level governs reset behavior. If the current login is admin, all parameters in the selected group (`All`, `Router`, or `Voice`) are reset. If the current login is user, then only the parameters in the selected group with user read-write permission are reset.

Finally, you also can perform a factory reset via remote provisioning. The [Device Configuration Profile Formats](#) section describes this method.

## **End-User ‘User’ Parameter Space**

A service provider can enable end-user control of a subset of the device parameters from the device 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 device 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 device 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 device firmware locally from the device web page or the IVR. The service provider may also update the firmware remotely via provisioning.

To get the latest firmware of OBi Voice Adapters, please login to your PDMS-SP account and visit the Firmware section.

To get a PDMS-SP account, please apply as a Polycom partner at <http://www.polycom.com/partners/become-partner.html> and notify your Polycom account representative (or contact [ITSPSales@polycom.com](mailto:ITSPSales@polycom.com)).

## Firmware Update Methods

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

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

## Update from the Device Web Page

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

## 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 Script](#) 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 HTTP/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://prov-server.myitisp.com/OBi30x-3-2-5997.fw
```

### Important Note:

- If the path contains the filename of the firmware file, the filename must be in the format shown in the above example.

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

#### Parameters That Control Firmware Update Using FirmwareURL

Parameter	Description
<code>X_DeviceManagement.FirmwareUpdate.Method</code>	<p>Choices are:</p> <ul style="list-style-type: none"> <li>• <code>Disabled</code> = Do not check for firmware upgrade from <b>FirmwareURL</b></li> <li>• <code>System Start</code> = Check for firmware upgrade from <b>FirmwareURL</b> on system start only</li> <li>• <code>Periodically</code> = Check for firmware upgrade from <b>FirmwareURL</b> on system start and also periodically at the interval specified in the <code>Interval</code> parameter</li> <li>• <code>Time of Day</code> = Check for firmware upgrade from <b>FirmwareURL</b> at certain time of day specified by the <code>TimeOfDay</code> parameter</li> </ul> <p>Note: The first firmware upgrade check on system start occurs after a random delay of 0 to 30 seconds.</p>
<code>X_DeviceManagement.FirmwareUpdate.Interval</code>	<p>When <code>Method</code> is set to <code>Periodically</code>, this parameter specifies the number of seconds between each checking of the firmware upgrade from <b>FirmwareURL</b>. If the value is 0, the device checks just once on system start only (equivalent to setting <code>Method</code> to <code>System Start</code>).</p>

**Parameters That Control Firmware Update Using FirmwareURL**

Parameter	Description
X_DeviceManagement.FirmwareUpdate.TimeofDay	<p>Time of day in hh:mm[+rr] format, valid when method is set to Time of Day. Choices are:</p> <ul style="list-style-type: none"> <li>• hh: 0 to 23</li> <li>• mm: 0 to 59</li> <li>• 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.</li> </ul> <p>Your devices always check once on system start, no matter what time it is. When NTP time is not available, your devices check for firmware updates every 60 minutes.</p>
X_DeviceManagement.FirmwareUpdate.FirmwareURL	<p>The basic syntax is a URL to download the firmware package. The full syntax is a provisioning script as described in the <a href="#">Provisioning Script</a> section. The supported schemes are http://, https://, and tftp://. For example:</p> <p style="padding-left: 40px;">http://prov-server.myitsp.com/OBi30x-3-1-2-5997.fw</p> <p>or</p> <p style="padding-left: 40px;">http://\$USR:\$PWD@prov-server.myitsp.com/6-3-0-16863/</p> <p>if authentication is required.</p> <p>Account information is stored in the <code>username</code> and <code>password</code> parameters.</p>
X_DeviceManagement.FirmwareUpdate.DnsLookup	<p>DNS record to use when resolve the hostname used in <b>FirmwareURL</b>. Choices are:</p> <ul style="list-style-type: none"> <li>• A Record Only</li> <li>• SVR Record Only</li> <li>• Try Both</li> </ul> <p>Note: When choosing Try Both, the SVR record is adopted if both records are available.</p>
X_DeviceManagement.FirmwareUpdate.DnsSrvPrefix	<p>Specifies whether the prefix is inserted automatically when performing a lookup for SRV record. Choices are:</p> <ul style="list-style-type: none"> <li>• NO prefix</li> <li>• With Prefix</li> <li>• Try Both</li> </ul> <p>Note:</p> <ul style="list-style-type: none"> <li>• <code>http._tcp.</code> or <code>tftp._udp.</code> is prepended to the name as a prefix, depending on the scheme used in <b>FirmwareURL</b>.</li> <li>• When choosing Try Both, the SVR record is adopted if both records are available.</li> </ul>
X_DeviceManagement.FirmwareUpdate.Username	<p>The account username if authentication is required when getting the firmware using http or https.</p>
X_DeviceManagement.FirmwareUpdate.Password	<p>The account password if authentication is required when getting the firmware using http or https.</p>

# Device Configuration Profile Formats

---

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

## Device Profile Format

Device profiles for device 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 devices.

### *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"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
  </xs:complexType>
</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:schema>

```

Referring to this 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
- **Router:** Factory reset router 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 device to perform a full system reboot after it completes processed the profile. This attribute should be sent to the device 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 device web page
- `readWrite`: User can only read and set the parameter value from local device web page
- `noAccess`: User can't see the parameter from local device 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 **parameter** attribute of the `<Value>` element specifies whether to use the default value for that parameter. If a non-empty 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 device 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 **parameter** attribute with valid values, if present, is still applied). Parameter values containing reserved XML characters such as

- `>` (0x3E)
- `<` (0x3C)
- `&` (0x26)
- `"` (0x22)
- `'` (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 &lt;= 3.1.2.5997 ) FWU -T=TPRM2
          http://server.myitsp.com/OBi30x-3-1-2-5997.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>

```

You can find samples of complete device profiles for each device at <https://www1.obihai.com/docs/>.

## Compact Profile Format

The device 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:

- <O> = <Object>
- <N> = <Name>
- <V> = <Value>
- <P> = <ParameterValueStruct>
- <X\_R> = X\_Reset
- <X\_UD> = X\_UseDefault
- <X\_UA> = X\_UserAccess
- "Y" = "Yes"
- "N" = "No"

Compact format and full format syntaxes can be mixed in the same profile.

## Profile Compression

To further reduce the size, you can compress profiles with gzip before sending to the devices. If the profiles are encrypted, encryption must be applied AFTER gzip compression.

## Device Parameters for Remote Provisioning

A number of parameters control how the device pulls a profile from the provisioning server. The following table summarizes these parameters. See the [Polycom OBi Device Administrator Guide](#) for a complete reference of available parameters.

### Parameters for Remote Provisioning

Parameter Name	Description
<b><i>X_DeviceManagement.ITSP Provisioning.Method</i></b>	<p>This parameter controls if and when the device should download the latest profile from the provisioning server. Choices are:</p> <ul style="list-style-type: none"> <li>• <code>Disabled</code> = Do not attempt to download profile</li> <li>• <code>System Start</code> = Download from <b>ConfigURL</b> just once on system start</li> <li>• <code>Periodically</code> = Download from <b>ConfigURL</b> on system start, and then periodically at the interval specified in the <code>Interval</code> parameter</li> </ul> <p>Note: The first download on system start is performed after a random delay of 30 to 90 seconds.</p>
<b><i>X_DeviceManagement.ITSP Provisioning.Interval</i></b>	<p>When method is set to <code>Periodically</code>, this is the number of seconds between downloads from the provisioned <b>ConfigURL</b>. If the value is 0, the device downloads just one time only on system start (equivalent to setting the method to <code>System Start</code>).</p>

## Parameters for Remote Provisioning

Parameter Name	Description
<b><i>X_DeviceManagement.ITSP Provisioning.ConfigURL</i></b>	In the simplest form, this can be just a URL to download the profile such as: <code>http://prov-server.myitsp.com/obi\$MAC.xml</code> The full syntax is a provisioning script as described in the <a href="#">Provisioning Script</a> section.
<b><i>X_DeviceManagement.ITSP Provisioning.SPRM0</i></b> to <b><i>X_DeviceManagement.Provi sioning.SPRM7</i></b>	Nonvolatile special parameters that can be used in a provisioning script and can be changed by provisioning only. The <b>SPRMn</b> values aren't accessible via the web management UI. These can be used in the <b>ConfigURL</b> as the option of SYNC Command only. Please see the <a href="#">Provisioning Script</a> section for details.
<b><i>X_DeviceManagement.ITSP Provisioning.GPRM0</i></b> to <b><i>X_DeviceManagement.ITSP Provisioning.GPRM7</i></b>	Nonvolatile general parameters that can be used in a provisioning script and can be changed by both the remote provisioning and web management interfaces.
<b><i>X_DeviceManagement.ITSP Provisioning.TPRM0</i></b> to <b><i>X_DeviceManagement.ITSP Provisioning.TPRM3</i></b>	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 <b>TPRMn</b> parameters are cleared.

## Provisioning Script

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 device 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>	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.



## Provisioning Script Parameters

Parameter Name	Description
IF	The string IF (0x49 0x46) which must be followed by (expr). This tells the device to execute the following operation in the statement only if the condition specified in expr is matched.
expr	<p>An expression enclosed in parentheses. The format of expr must be</p> <pre>\$MacroName &lt;SP&gt; ComparisonOperator &lt;SP&gt; Value.</pre> <p>Where:</p> <p>MacroName is the name of a defined macro, such as TPRM0 or MAC.</p> <p>Value can be a combination of ASCII string and macros, or a quoted string.</p> <p>Examples:</p> <ul style="list-style-type: none"> <li>• Abcde</li> <li>• \$DM\$MAC</li> <li>• abcde\${MAC}.xml "abc def hijk"</li> </ul> <p>It may not contain any &lt;SP&gt; characters except when enclosed by double quotes. The enclosing double quotes are excluded when doing the comparison.</p> <p>ComparisonOperator is one of the following relational operators:</p> <ul style="list-style-type: none"> <li>• == (Equal)</li> <li>• != (Not equal)</li> <li>• &gt;= (Larger than or equal to)</li> <li>• &gt; (Larger than)</li> <li>• &lt;= (Less than or equal to)</li> <li>• &lt; (Less than)</li> </ul> <p>The definition of the relational operation follows that of the standard C library function strcmp(char *str1, char *str2).</p>
Oper	<p>One of the following operations: SYNC, FWU, WAIT, EXIT, GOTO, SET, CLR.</p> <p>The <a href="#">Provisioning Script Operations</a> section describes these operations.</p>

## 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 device's profile with a profile specified by the URL. The device 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: <pre>000102030405060708090a0b0c0d0e0f</pre> 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 device 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 <pre>00102030405060708090a0b0c0d0e0f0</pre> iv is not needed for RC4. It is optional for AES. If not specified, the device 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 device 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 device 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 device 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 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 currently stored on the device.

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
```

## FWU (Firmware Update)

This operation lets the device 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 TPRM $x$`  to store the result, where  $x = 1, 2, \text{ or } 3$ . By default, the result is stored in `TPRM0`.
- `URL = URL` to download the firmware. HTTP and TFTP schemes are supported.

In the context of the **FirmwareURL** parameter, the opcode FWU is implied if omitted.

Example:

```
IF ( $FWV <= 3.1.2.5997 )
FWU http://server.myitsp.com/OBi30x-3-1-2-5997.fw
```

In this example, the device is updated to the firmware at the given URL only if the current firmware version is older than 3.1.2.5997.

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 device 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 device 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 device 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 device 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't 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 doesn't 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 malformed
- 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.myitisp.com/OBi30x-3-1-2-5997.fw;
```

Note that the FWU opcode is implied in this simple case.

### ***Example 2: (ConfigURL) Sync to a Specific Profile***

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

Note that the SYNC opcode is implied in this simple case.

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

The device updates to firmware version 3.1.1.5589 if its current version is older than that. Otherwise it updates to firmware version 3.1.2.5997 if its current version is older than that. In case the upgrade fails, the device retries in 60 seconds. Note that the device reboots if it updates to version 3.1.1.5589. On bootup, it runs the same script again and updates to 3.1.2.5997.

```
@start SET TPRM2 = 2;
  IF ( $FWV < 3.1.1.5589 ) FWU -T=TPRM2
  http://prov-server.myitisp.com/OBi30x-3-1-1-5589.fw;
  IF ( $TPRM2 == 1 ) EXIT;
  IF ( $TPRM2 == 0 ) GOTO error;
  IF ( $FWV < 3.1.2.5997 ) FWU -T=TPRM2
  http://prov-server.myitisp.com/OBi30x-3-1-2-5997.fw; IF ( $TPRM2 != 0 ) EXIT;
@error WAIT 60;
  GOTO start;
```

### ***Example 4: (ConfigURL) Download with Two Profiles Sequentially***

The device 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 device 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 device (**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 the Ready threads to run. If there are more than one Ready threads, the **FirmwareURL** script has priority over the **ConfigURL** script.

The device'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 device, one for ITSP provisioning and one for OBiTALK provisioning. The ITSP provisioning **ConfigURL** script has higher priority over the OBiTALK provisioning **ConfigURL** script.

## Device Behavior on Processing a Profile

As soon as the device 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 device 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 Script](#) 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 device 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`

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 device 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***

Remote provisioning relies on the device to initiate downloading of the profile. A simple mechanism for the service provider to force the device 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 device with Event header set to `Reboot`. The device 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 device 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 device behind a firewall, or by enabling the device to challenge the request with the same user-id and password provisioned on that user account.

## ***Firewall Considerations***

Most devices sit behind a firewall. Normally it is not possible to send down an unsolicited SIP NOTIFY request to the device, unless the device is registered with the service provider and gets through the firewall to allow the request to get in. Registration is done periodically by the device with an interval specified by the service provider. The interval can be set small enough so that the access remains open to the service provider between registration renewals. Note that the firewall access is only available to the server where the registration is sent.

## **Creating Profiles for Deployment**

There are many ways to create a device profile. The choice largely depends on your work flow and the available tools with which you are comfortable. 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 device can be used by a different end user with different credentials, the final deployment profile for each device is different. However, most of the configuration parameters in the profile are the same for all devices. 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 device.

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 device for the first time, and subsequently include a small subset of parameters in the day-to-day profile.

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. You can use the **X\_Reset** attribute in the

<ParameterList> (root) element in a profile to force the device to do a one-time factory reset of all parameters (refer to the [Device Profile Format](#) section).

If you have any questions, ask for assistance from [Obi.SP.Support@Polycom.com](mailto:Obi.SP.Support@Polycom.com). Polycom support is here to help to make your deployment a success.

## Backing Up a Profile from the Device Web Page

The device'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 device'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.



All passwords and PINs (all values that are masked on the device web page) are excluded from the backup file.

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

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

## Use the PDMS-SP Cloud Management Portal

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 devices being managed by the service provider. From there, the administrator can also add more devices to the service provider account to be managed. The administrator can click the **Device ID** of any of the managed devices to view and change its configuration.

To generate a profile for a particular device model, add at least one device of that model to the service provider account. Then click the **Device ID** of that device on the **PDMS-SP portal** to reach the **Manage Device** page for that device. Then make changes to the device parameters by clicking the **Goto Device Configuration** button. The page layout is similar to the local device 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 device 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 device (see the previous 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 device web page or the ITSP portal's device 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 devices, certificate management, please go to <http://pki.polycom.com> and follow the instructions.

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

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

## ***Using HTTPS***

The most secure way available to download a device profile from the provisioning server is by using HTTPS. With HTTPS, both the device and provisioning server can verify the identity of each other. The data exchanged between the device 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 device 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 device is authentic: that the device 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 email to [OBi.SPSupport@polycom.com](mailto:OBi.SPSupport@polycom.com) and copy your Polycom sales representative.

Polycom also provides a web configuration template generator tool to help you search for parameters and generate an XML out of it. Please login to your PDMS-SP portal, go to the Tools, and find the Configuration Template Generator section.

To get access to PDMS-SP, please apply as a Polycom partner at our website:

<http://www.polycom.com/partners/become-partner.html> and email [ITSPSales@polycom.com](mailto:ITSPSales@polycom.com) to request access.

## Server Authentication

During HTTPS handshake, the device 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 device must have the CA certificate that signs the server's certificate in its verification chain. Currently the devices 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=IE, O=Baltimore, OU=CyberTrust, CN=Baltimore CyberTrust Root
- 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 device 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](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 device.

The devices support two cryptos for profile encryption: AES128 (CBC with PKCS#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 00102030405060708090a0b0c0d0e0f0 -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 device. This is the OBi Default Encryption method, where the encryption algorithm is proprietary and the secret key is derived by each device 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 device's 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 device with the same MAC address and checking the contents from the device web page.

It should be advised that the device'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 device 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 device by initially provisioning it once with HTTPS. It is recommended to store the secret key in one of the `SPRMx` parameters, and reference it in the **ConfigURL** with its corresponding `$SPRMx` macro.

## Automating Device 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 device attempts to download a generic profile once when the system boots up. The **\$DHCPOPT66** macro expands into option 66 offered by the (local) DHCP server. If DHCP is disabled on the device 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 device executes the **ConfigURL** and downloads the `$DM.xml` profile. The macro `$DM` is expanded into the model name of the device. 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 device at the last step. The last profile is also the day-to-day profile that the device regularly grabs from the field.

As the first profile in the chain, `$DM.xml` may contain a few parameters to establish some basic boundaries for the device 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.myitisp.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">
```



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 device is running the firmware version of your choice. You can do this by inserting a proper **FirmwareURL** in `$DM.xml`.

In practice, the device 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 device web page and checking the **ConfigURL**, for instance). You could also wait until the last profile is loaded onto the device before shipping it out, verifying 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 device). 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 `$MAC-init.xml` is to store a secret decryption key in the device and to let the device subsequently switch to use the encrypted profile. As shown in the listing `$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 device, hence the need to include `$MAC` in the profile name so that the server can tell which device 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.myitisp.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`.

## Example Profile Listings

**\$DM.xml** (Replace `$DM` with the device name, such as `OBi202`)

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Generic Configuration File (OBi202.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 ( $FWV &lt; 1.0.3.1890Bi202-1-1-0-1891.fw </Value>
    </ParameterValueStruct>
  </Object>
  <Object>
    <Name>X_DeviceManagement.ITSPProvisioning.</Name>
    <ParameterValueStruct>
      <Name>Method</Name>
      <Value>Periodically</Value>
    </ParameterValueStruct>
    <ParameterValueStruct>
      <Name>Interval</Name>
      <Value>3600</Value>
    </ParameterValueStruct>
    <ParameterValueStruct>
      <Name>ConfigURL</Name>
      <Value> SYNC https://prov-server.myitisp.com/$MAC-init.xml </Value>
```

```

    </ParameterValueStruct>
  </Object>
</ParameterList>

```

### **\$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>
<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 ( $FWV &lt; 1.0.3.1891 ) FWU http://prov-server.myitsp.com/VVX250-1-1-0-1891.fw
    </Value>
  </ParameterValueStruct>
</Object>
</Object>
<Object>
  <Name>X_DeviceManagement.Provisioning.</Name>
  <ParameterValueStruct>
    <Name X_UserAccess="noAccess">Method</Name>
    <Value>Periodically</Value>
  </ParameterValueStruct>
  <ParameterValueStruct>
    <Name X_UserAccess="noAccess">Interval</Name>
    <Value>3600</Value>
  </ParameterValueStruct>
  <ParameterValueStruct>
    <Name X_UserAccess="noAccess">ConfigURL</Name>
    <Value>
SYNC -A=aes -K=$SPRM0 -IV=$SPRM1 http://prov-server.myitsp.com/$MAC-encrypted.cfg
    </Value>
  </ParameterValueStruct>
</Object>
</Object>
<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>
```