TECHNICAL REFERENCE GUIDE

7.1.0 | July 2020 | 3725-49221-001A

VVX D230 DECT IP Phone

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

Before You Begin

This guide includes technical reference information for administering, configuring, and provisioning VVX D230 systems, focusing on the D230 DECT IP phone base station. For information specific to configuring features on the VVX D230 DECT IP phone handset, see the *Poly VVX D230 DECT IP Phone Administrator Guide*.

Audience, Purpose, and Required Skills

This guide is for a technical audience. You must be familiar with the following concepts before beginning:

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

Related Poly and Partner Resources

See the following sites for information related to this release.

- The Poly Online Support Center is the entry point to online product, service, and solution support information including Licensing & Product Registration, Self-Service, Account Management, Product-Related Legal Notices, and Documents & Software downloads.
- The Polycom Document Library provides support documentation for active products, services, and solutions. The documentation displays in responsive HTML5 format so that you can easily access and view installation, configuration, or administration content from any online device.
- The Poly Community provides access to the latest developer and support information. Create an account to access Poly support personnel and participate in developer and support forums. You can find the latest information on hardware, software, and partner solutions topics, share ideas, and solve problems with your colleagues.
- The Poly Partner Network are industry leaders who natively integrate the Poly standards-based RealPresence Platform with their customers' current UC infrastructures, making it easy for you to communicate face-to-face with the applications and devices you use every day.
- The Polycom Collaboration Services help your business succeed and get the most out of your investment through the benefits of collaboration.

System Configuration Parameters

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

- Canonical fashion

- Literal fashion

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

The canonical fashion simplifies locating parameters on your device's system web interface, the PDMS-SP service, or in the Polycom Obihai Data Model, which is a collective list of configuration parameters, syntaxes, and valid values for VVX D230.

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's system web interface or PDMS-SP Configuration web page. This string may contain spaces. When a group heading has more than one level, a – separates each level, such as:

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

The `ParameterName` is the name of the parameter as shown on the web page and mustn't contain any spaces. ***Parameter Group Name*** and `ParameterName` are separated by two colons (::), as shown in the first example.

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

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

Literal Fashion

These examples show the format of the literal fashion. Use the literal fashion when provisioning systems.

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

The ***ParameterGroupName.*** is the name of the first parameter group in literal fashion. This string mustn't contain any spaces, and always ends with a period, as shown. You can use more than one ***ParameterGroupName.*** The ***ParameterGroupName.*** is case-sensitive.

The **ParameterName.** is the name of the parameter, and always ends with a period, as shown. This string mustn't contain any spaces. The **ParameterName.** is case-sensitive.

The `Parameter Value` is the literal value to assign to the named parameter and can 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 ***ParameterGroupName***.**ParameterName**.`Parameter Value`, but text formatting such as bold face isn't required and is removed when your script or app is processed.

Boolean Values

Parameters that take a Boolean value have a check box next to the parameter name on your system web interface pages. Throughout the document, we refer to a Boolean value as “enable or disable” or “yes or no”, but the only valid Boolean parameter values to use in a phone configuration file is either `true/false` or `True/False` (case-sensitive). This is equivalent to selecting or clearing the check box on the configuration web pages.

Multiple Choice Values

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

Parameter Values

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

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

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

For example: **X_STUNServerPort** `= UDP listen port of the STUN Server`

XML Usage

When you write or edit XML for your VVX D230 system, use an XML editor that automatically checks your syntax.

Call Routing

Call routing rules are parameters that instruct your system how to route calls. A call can transform into a call bridge or an endpoint call after being routed by your system according to the given routing rules. The call routing rule syntaxes for inbound calls and outbound calls are slightly different.

Call routing rule configuration relies heavily on digit maps. If you aren't familiar with how digit maps work, see Digit Map Configuration.

Inbound Call Route Formatting and Rules

The general format is:

`InboundCallRoute:= rule` OR `{rule},{rule},….`

You can omit the curly braces if your route contains only one rule. The OR operator isn't part of the parameter syntax. Here, it separates alternative values.

A rule has the following format:

```
rule := peering-list : terminal-list
```

The following table shows the rule formats.

Rule Formats

Rule	Format	Notes
`peering-list :`	`peering,peering,…`	Comma-separated list of 0 or more peering objects.
`terminal-list :`	`terminal,terminal,….`	Comma-separated list of 0 or more terminal objects.
`peering :`	`caller-list > callee-list`	
`caller-list :`	`caller|caller|caller|…`	Vertical bar-separated list of 0 or more caller objects.
`callee-list :`	`callee|callee|callee| …`	Vertical bar-separated list of 0 or more callee objects.
`caller :`	`number` OR `embedded-digit-map` OR `?` OR `@`	`?` = anonymous, `@` = any number but anonymous.
`callee :`	`number` OR `embedded-digit-map` OR `@`	
`terminal :`	`SPx(arg)` OR `PPx(arg)`	`arg` object is optional.
`arg :`	`cid > target`	

Rule Formats (continued)

Rule	Format	Notes
`x :`	`1` OR `2` OR `3`…	Where applicable. Can be omitted if x = 1.
`cid :`	`spoofed-caller-number` OR `$1`	
`target :`	`number-to-call` OR `$2`	
`embedded-digit-map :`	`(Mlabel)` OR `digit-map`	

General notes:

- `terminal-list` can be empty, which means to block this call. The preceding ':' can't be omitted. You can specify as many as four terminals in the list. The listed terminals are called/rung by your system simultaneously. This operation is known as forking the call. A terminal can be a trunk or an endpoint.
- Abbreviated terminal names are case-insensitive.
- `number` and `number-to-call` are literal strings, such as `14089991234`.
- `digit-map` is just any proper digit map, such as `(1xxx|xx.)`. Make sure to include the enclosing parentheses.
- `spoofed-caller-number` is a literal string, such as `14081112233`, to be used as the caller number for making a new call on the specified trunk.
- `(Mlabel)` is a named digit map, where `label` is the abbreviated name of any terminal that has a digit map defined: `SP1`, `SP2`, `SP3`, `SP4`, `SP5`, `SP6`, `SP7`, and `SP8`.
- `$1` is an internal variable containing the value of the caller number of this inbound call, after any digit map transformation in the matched caller object of the matched peering object in the `peering-list`.
- `$2` is an internal variable containing the called number of this inbound call, after any digit map transformation in the matched callee object of the matched peering object in the `peering-list`.

Notes on `peering-list` and `peering` objects:

- `peering-list` is optional in `InboundCallRoute`. If `peering-list` is empty, the succeeding ':' can be omitted also. An empty `peering-list` implies a single peering object whose caller object list matches any caller number. The following `InboundCallRoute` are all equivalent:
  - `dt1`
  - `{dt}`
  - `{:dt}`
  - `{?|@>@:dt}`
- `callee-list` in a peering object can be empty. It implies the callee object `@`, meaning any called number. The preceding '>' can be omitted if `callee-list` is empty.
- `caller-list` in a peering object can be empty. It implies the caller-list `@|?`, meaning any caller number including anonymous. The succeeding '>' can't be omitted if `caller-list` is empty but not the `callee-list`.

Notes on the `arg`, `cid`, and `target` objects:

- The `cid` object inside an `arg` object is optional. If omitted, it implies no caller-ID spoofing when making the call on the specified trunk. The succeeding '>' can be omitted if `cid` is omitted.
- The `target` object inside an `arg` object is optional. If omitted, it implies the target `$2`, which means to call the original called number after applying any necessary digit map transformation implied by the rule. The preceding '>' can't be omitted if `target` is omitted but `cid` isn't.
- The `arg` object is optional. If omitted, it implies the `arg` with the target `$2` and no `cid`. If `arg` is omitted, the succeeding parentheses `()` can be omitted also.

An inbound call matches a rule if its caller-number/callee-number matches one of the peering objects of the rule. Peering objects are tested in the order left to right, and the first matched peering object wins. Rules are also checked in the order left to right, and the first matched rule wins. Therefore, you need to place the more specific rules first in the `InboundCallRoute` if multiple rules can potentially match the same inbound call.

Inbound Call Route Examples

The following are some examples of inbound call route rules and formats:

`dt` OR `{dt}` OR `{:dt}` OR `{@|?>@:dt}` (all equivalent)

It says: Ring the handset for all incoming calls. This is the default `InboundCallRoute` for all trunks.

```
{(14081223330|15103313456):aa},{(1800xx.|1888xx.):},{dt}
```

It says: Ring both handset and AA for calls coming from `1 408 122 3330` or `1 510 331 3456`, block all `800`, `888`, and anonymous calls, and ring the handset for all other calls.

```
{(x.4081113333|x.4152224444):aa},{dt}
```

It says: Ring the AA for calls coming from any number that ends with `408 111 3333` or `415 222 4444`, and ring the handset for all other calls. Be sure to include the enclosing parentheses in this example, since "`x.`" is a digit map specific syntax.

```
{200123456:aa},{sp1(14083335678)}
```

It says: Ring the AA for calls coming from `200123456`. For all any other call, bridge it by calling `1 408 333 5678` using SP1 Service.

Outbound Call Route Rules and Formatting

The general format is:

`OutboundCallRoute`:= `rule` OR `{rule},{rule},`….

You can omit the curly braces if the route contains only one rule. The OR operator isn't part of the parameter syntax. Here, it separates alternative values.

A rule has the following format:

```
rule := callee-list : terminal
```

where

- `callee-list:= callee|callee|callee|` …(vertical bar separated list of 0 or more callee objects)

- `callee:= number` OR `embedded-digit-map` OR `@` (`@` = any number)
- `terminal:= DTx OR SPx(arg)` OR `PPx(arg)` (`arg` object is optional)
- `arg:= cid > target`
- `x:= 1` OR `2` OR `3`…(where applicable. Can be omitted `x` = 1.)
- `cid = spoofed-caller-number`
- `target = number-to-call` OR `$2`
- `embedded-digit-map = (Mlabel)` OR `digit-map`

General notes:

- A terminal can be a trunk or another endpoint.
- Abbreviated terminal names are case-insensitive.
- `number` and `number-to-call` are literal strings, such as `14089991234`.
- `digit-map` is just any proper digit map, such as `(1xxx|xx.)`. Make sure to include the enclosing parentheses.
- `spoofed-caller-number` is a literal string, such as `14081112233`, used as the caller number for making a new call on the specified trunk.
- `(Mlabel)` is a named digit map where label is the abbreviated name of any terminal that has a digit map defined: SP1, SP2, LI, PP, or DT.
- `$2` is an internal variable containing the called number of this outbound call, after any digit map transformation in the matched callee object.
- `callee-list` can be empty, which implies the single callee object `@`, which means any called number. The succeeding '`:`' can be omitted also when callee-list is empty.

Notes on the `arg`, `cid`, and `target` objects:

- The `cid` object inside an `arg` object is optional. If omitted, it implies no caller-ID spoofing when making the call on the specified trunk. The succeeding '`>`' can be omitted if `cid` is omitted.
- The `target` object inside an `arg` object is optional. If omitted, it implies the target `$2`, which means to call the original called number after applying any necessary digit map transformation implied by the rule. The preceding '`>`' can't be omitted if `target` is omitted but not `cid`.
- The `arg` object is optional. If omitted, it implies `arg` with the target `$2` and no `cid`.

An outbound call matches a rule if its called number matches one of the callee objects of the rule. Callee objects are tested in the order left to right, and the first matched callee wins. Rules are also checked in the order left to right, and the first matched rule wins. Therefore, you need to place the more specific rules first in `OutboundCallRoute` if multiple rules can potentially match the same outbound call.

Note that every endpoint also has a digit map defined. The user-dialed number is completely processed with the endpoint's digit map first before passing it to `OutboundCallRoute` for a routing decision. Therefore the number used for matching call routing rules has already incurred the transformations, if any, implied by the digit map. Remember this fact when crafting your own `OutboundCallRoute`.

OutboundCallRoute Examples

The following are some examples of inbound call route rules and formats:

`sp1` OR `{SP1}` OR `{:SP1}` OR `{@:Sp1}` (all equivalent)

This rule says: Make all calls using the SP1 Service, without any caller-id spoofing or digit transformation.

`{(Mpli):pli},{(<**1:>(Msp1)):sp1},{(<**2:>(Msp2)):sp2},{(<**8:>(Mli)):li},{(<**9:>(Mpp)):pp}`

This is the default `OutboundCallRoute` for the handset. It says:

- Dial *** to invoke the local system configuration IVR.
- `(Mpli)` and `pli` are substituted with the Primary Line's abbreviated name.
- Use SP1 Service to call all numbers that start with **1 and subsequent digits matching SP1 Service's `DigitMap`. Remove the **1 prefix from the resulting number before making the call.
- Use SP2 Service to call all numbers that start with **2 and subsequent digits matching SP2 Service's `DigitMap`. Remove the **2 prefix from the resulting number before making the call.
- Use the PDMS-SP Service to call all numbers that start with **9 and subsequent digits matching PDMS-SP Service's `DigitMap`. Remove the **9 prefix from the resulting number before making the call.

Digit Map Configuration

A digit map serves to transform and restrict the number that can be dialed or called, and determine if you dialed sufficient digits to form a complete number. Each map is composed of one or more rules surrounded by parentheses that MUST NOT be omitted. Here is the general format of a digit map:

`(rule|rule|….|rule)`

A digit map rule is a rule for matching a given sequence of digits. It can contain extra white spaces for readability. All spaces are removed by your system during parsing. A rule can contain one or more of the following elements:

- `literals` – Any combination of `0-9`,`*`,`#`,`+`,`-`,`A-Z`,`a-z`, except `m`, `M`, `s`, `S`, `x`, and `X`, which have special meaning in the digit map syntax. It matches digit sequences with exactly the same literals.
- `'literals'` – Everything inside a pair of single quotes is treated as a literal except for the single quote (') character.
- `x` – a wild card digit that matches any digit from `0-9`. `x` is case-sensitive.
- `x.` – matches 0 or more `x`.
- `[123-7]` or `[135]` – A set of 1 or more digits surrounded by pair of `[ ]`. It matches any digit in the set. The `-` syntax represents an inclusive digit range, such as `0-9`, `3-7`. So `[123-7]` is equivalent to `[1-7]` or `[1234567]`.
- `S`, `S0`, `S1`, `S2`, …`S9` – Digit timer of 0, 1, 2, …,9 seconds. `S` is equivalent to `S1`. `S0` is the same as "blank". You can concatenate multiple `S` elements together if you need more than 9 seconds timeout, such as `S9S5` for a 14-second timeout.

  `S` is case-sensitive. It should only be used either as the first element of a rule for hot/warm line implementation, or as the last element of a rule as a means of overriding the default interdigit timer.

- `<elements:literals>` – Substitute the digit sequence matching elements with the given literals. Single quote syntax isn't needed or allowed for the literals in this context. Special characters can be used here as they don't apply in this context either. Elements can be empty, in which case the '`:`' can be omitted. This case is useful for inserting some extra digits in certain part of the dialed digits. The literals part can be empty also but the '`:`' MUST NOT be omitted. This case is useful for removing part of dialed digits. Elements and literals MUST NOT be both empty.
- `(map)` – An embedded digit map for matching subsequent digits.
- `(Mlabel)` – A named embedded digit map for matching subsequent digits, where label is one of abbreviated terminal names. Possible choices are:
  - `(Msp1)` for ***SP1 Service::***`DigitMap`
  - `(Msp2)` for ***SP2 Service::***`DigitMap`
  - `(Msp3)` for ***SP3 Service::***`DigitMap`
  - `(Msp4)` for ***SP4 Service::***`DigitMap`
  - `(Msp5)` for ***SP5 Service::***`DigitMap`
  - `(Msp6)` for ***SP6 Service::***`DigitMap`
  - `(Msp7)` for ***SP7 Service::***`DigitMap`
  - `(Msp8)` for ***SP8 Service::***`DigitMap`
  - `(Mpp)` for ***OBiTALK Service::***`DigitMap`

Starting with release 1.2, the following elements are added:

- `X` – A wildcard digit that matches 0–9 or *. This is equivalent to `[x*]` or `[0-9*x]`
- `@` – A wildcard character that matches any alphanumeric character except `#`
- `x?` – matches 0 or 1 `x`
- `@?` – matches 0 or 1 `@`
- `[^…]` – matches any single alphanumeric character that isn't in the set
- Allow alphanumeric and wildcard characters inside a set `[ ]`, such as `[x]`, `[X#]`, `[@#]`, `[a-zA-Zx]`

The last two elements imply that your system's digit maps are recursive. Recursive digit maps allow digit maps to be reused and make their specification more compact and readable. It's important that you don't specify digit maps that lead to infinite recursion. For example, a digit map must not include a named embedded digit map that references itself.

To bar users from calling numbers that match a rule, add a '`!`' in front of that rule in the digit map. The rule is then referred to as a barring rule.

Examples:

- `1408xxxxxxx` – Matches any 11-digit number that starts with `1408`.
- `011xx.` – Matches any number that starts with `011` followed by one or more digits.
- `<1408>xxxxxxx` – Matches any 7-digit number. Your system prepends `1408` to the number when making the call.
- `<:1408>xxxxxxx` – Equivalent to the last example.
- `<+>1xxxxxxxxxx` – Prepends '`+`' to any 11-digit number that starts with `1`.
- `<**1:>1408xxxxxxx` – Matches any number that starts with `**11408` followed by 7 digits. Your system removes the `**1` prefix when making the call.
- `*74(x|xx)` – Matches any number that starts with `*74`, followed by 1 or 2 digits.

- `**1(Msp1)` – Matches any number that starts with `**1` and with the rest of digits matching the `DigitMap` in the SP1 Service.
- `<:1234>` – Matches an empty phone number and replaces with `1234`. This is the syntax for a hotline to `1234`.
- `<S0:1234>` – Equivalent to the last example.
- `<:#>` – Hotline to the number `#`.
- `<S0:#>` – Equivalent to the last example.
- `<S4:1234>` – Call `1234` if no digits entered for 4 seconds. This is the syntax of a warm line.
- `xx.853 7683` – Matches any number with at least 8 digits and ends with `8537683`, such as `15108537683`, `98537683`.
- `(x.408 223 1122)` – Matches any number with at least 10 digits and ends with `408 223 1122`, such as `4082231122` or `1408 223 1122`.
- `xx.<#>` – Adds a `#` to the end of any number with 1 or more digits.
- `!1900xxx xxxx` – Barring all 11-digit numbers that start with `1900.`
- `[^*]@@.` – Arbitrarily long alphanumeric sequence (except `#`) that doesn't start with `*`
- `xx?` – Any 1- or 2-digit number.
- `(1xxxxxxxxxxS0|xx.)` – Arbitrarily long digit sequence not starting with `1`. Otherwise it's limited to 11 digits.

Match Against Multiple Rules in a Digit Map

One important function of a digit map is to determine if you dialed sufficient digits during dialing. A digit map normally contains more than one rule. The Digit Map Processor (DMP) must return the best matched rule at some point, or declare that the input digit sequence is invalid. The DMP keeps refining its decision as each digit is entered until it reaches a final decision, or is forced to make a timely decision when the interdigit timer expires.

The DMP restarts the interdigit timer on every newly entered digit. The duration of this timer can be either long or short. The long and the short timer values are set by default to 10 seconds and 2 seconds, respectively, and are configurable per handset via the `DigitMapLongTimer` and `DigitMapShortTimer` parameters. Whether to use the long or short interdigit timer depends on the current rule matching states. The DMP maintains a matching state for each rule in the digit map as it processes each input digit. The following states are defined:

- Exactly Matched (EM) – The rule exactly matches the accumulated input sequence. However, any further input digit turns this rule into the MM state. Example: `1234` exactly matches the rules `xxxx`, `1234`, `1xxx`, `<123:5678>x`.
- Indefinitely Matched (IM) – The rule matches the accumulated input sequence indefinitely, with a variable length such that the rule can potentially stay as IM as more matching digits are entered. Example: 011853 indefinitely matches the rules `xx.`, `011xx.`, `<011:>xx`.
- Partially Matched (PM) – The rule partially matches the accumulated input sequence. Initially all rules are in this state before any digit is entered. Rules in this state have the potential of becoming EM or IM as more digits are entered. Example: `1234` partially matches the rules `xxxxxxx`, `1xxxx`, `1234567`, `<123:>xxxx.`
- Mismatch (MM) – The rule doesn't match the accumulated input sequence. This state won't change as more digits are entered. Example: `1234` mismatches the rules `123`, `1xx`, `12345`.

Rules in the EM or IM state are candidates to be selected by the DMP. After processing a new digit, the DMP returns a final decision if any of the following conditions holds:

- All rules are the MM state. The DMP returns an error.
- One or more rules are in the EM state with no rules in the IM state. DMP returns the best matched EM rule. If the best matched rule is a barring rule, DMP returns an error instead.

Otherwise, DMP starts the short interdigit timer if there is at least one rule in the EM state, or else the long one. When the interdigit timer expires, DMP returns the best matched rule at that moment if one is found, or else a timeout error. If the best matched rule in this case is a barring rule, DMP returns an error instead. Note that the timer to wait for the first input digit isn't governed by the interdigit timer, but the duration of dial tone being played, and could be a lot lengthier than the long interdigit timer.

The best matched rule is the one that has the most specific literals matching the input digit sequence. For example, the input sequence `1234` matches the rule `123x` better than `1xxx`. Note that an EM rule is always selected over an IM rule.

Finally, the default interdigit timer can be overridden by appending the `Sn` element at the end of the rule (*n* = 0–9).

Consider this simple digit map:

```
(<1408>xxx xxxx)
```

As soon as you enter 7 digits, the DMP returns a complete number by prepending the accumulated digits with `1408`.

Consider another simple map:

```
(xx.)
```

After you dial one or more digits, the DMP returns the accumulated digits as a complete number when the long interdigit timer expires.

Combine the last two maps:

```
(xx. | <1408>xxx xxxx)
```

After you dial one or more digits (but fewer than seven digits), the DMP returns the accumulated digits as a complete number when the (long) interdigit timer expires. As soon as seven digits are entered, the DMP returns `1408` followed by the accumulated seven digits when the (short) interdigit expires. On the eighth digit and beyond, however, the DMP considers the first rule only and returns the accumulated digits as-is when the (long) interdigit timer expires.

Now add an `S4` timer to the second rule:

```
(xx. | <1408>xxx xxxxS4)
```

In this case, the DMP behaves exactly the same as the last, except that the short interdigit timer the DMP uses upon receiving the seventh digit is overridden by a 4-second timer. Thus you've as long as 4 seconds instead of 2 to dial the eighth digit.

Example of Forcing an Interdigit Timeout with a Pound(#) Key

When dialing, you can force an interdigit timeout with a # key instead of waiting for the DMP to time out its own long or short timer. This is allowed as long as the # key doesn't match the current element of any PM rules. Otherwise the DMP consumes the # key instead of triggering a timeout.

Consider the digit map `(33xx.)`

If you enter `333#`, the DMP immediately returns the number `333`.

Now consider the digit map `(33xx.|333#1234x.)`

If you enter `333#`, the DMP won't return, but continues to wait for further input or for its interdigit timer to expire. Note that the first rule "`33xx`." is now in the MM state since the digit `#` doesn't match "`x`". You can continue to enter `1234#`, or `1234` and wait for a long interdigit timeout for the DMP to successfully return `333#1234`.

Invoking a Second Dial Tone in a Digit Map

You can tell your system to start a tone after a certain pattern of digits have been dialed by specifying the element `{t=<tone>}` within a digit map, where `<tone>` is a 1- to 3-letter name of the tone to play. The tone stops when you enter the next digit. For example:

```
(**1{t=di2}(Msp)|**8{t=od}(Mli))
```

tells your system to play Second Dial Tone when you dial **1, or play Outside Dial Tone when you dial **8. The acceptable (case-insensitive) values of `<tone>` are:

- `bu` = Busy Tone
- `cf` = Call Forwarded Dial Tone
- `cm` = Confirmation Tone
- `co` = Conference Tone
- `cw1 – cw10` = Call Waiting Tone 1-10
- `di` = Dial Tone
- `di2` = Second Dial Tone
- `fb` = Fast Busy Tone
- `ho` = Holding Tone
- `od` = Outside Dial Tone
- `pr` = Prompt Tone
- `rb` = Ringback Tone
- `ro` = Reorder Tone (same as fast busy)
- `si1 – si4` = SIT TONE 1 – 4
- `st` = Stutter Tone
- `0 – 9`, `*`, `#`, `a – d` = DTMF 0 – 9, *, #, A – D

Changing an Interdigit Long Timer Dynamically After a Partial Match

Your system starts off with the interdigit long timer set to the configured `DigitMapLongTimer` value when processing a new digit sequence by a digit map. You can change the long timer as some patterns are partially matched by embedding the syntax `{L=<time>}` within a rule in the digit map, where `<time>` is the desired number of seconds for the long timer. For example:

```
(011 853 xxxx xxxx{L=5}x. |xx.)
```

Here the long timer is shortened to 5 seconds after you enter `011 853` + 8 digits. Hence, your system declares that a complete number is collected in 5 seconds when it doesn't receive any more digits. Without the `{L=5}` syntax, you have to wait for 10 seconds (by default) for the same to happen.

User-Defined Digit Maps

The ***User Settings – User Defined Digit Maps*** section of the system web interface provides for 10 user-definable digit maps. These digit maps are referred to as User-Defined Digit Map 1 to 10. Each user-defined digit map is specified with two parameters:

- `Label`: An arbitrary string for referencing this digit map in other digit map specification. The value should be 2 to 16 characters long. For example, "friends". In this case, `(Mfriends)` can be referenced in other digit maps.
- `DigitMap`: A digit map to restrict the numbers that can be dialed or called with this service. Maximum length is 511 characters.

Both parameters are empty by default, except for ***User Defined Digit Map 1***.

Example of a User Defined Digit Map For IPv4 Dialing

The default values of the parameters for ***User-Defined Digit Map 1*** are set to the following values to support IPv4 dialing:

- `Label`: `ipd`
- `DigitMap`:
  `(xx.<*:@>xx?x?<*:.>xx?x?<*:.>xx?x?<*:.>xx?x?|xx.<*:@>xx?x?<*:.>xx?x?<*:.>xx?x?<*:.>xx?x?<*::>xx?x?x?x?)`

The map (`Mipd`) is referenced in the default setting of the `DigitMap` parameters in ITSP Profiles A through H. It supports the following two forms of IPv4 dialing:

- `<user-id>*<a>*<b>*<c>*<d>`
- `<user-id>*<a>*<b>*<c>*<d>*<port>`

where `<user-id>` is an arbitrary length numeric user-id, such as `100345`, `<port>` is a port number in the range 0–65535, and each of `<a>`,`<b>`,`<c>`,`<d>` is a 1- to 3-digit pattern in the range 1–255 that identifies one byte of an IP address. The dialed number is translated into `<user-id>@<a>.<b>.<c>.<d>` and `<user-id>@<a>.<b>.<c>.<d>:<port>`. Here are some examples:

- `1234*192*168*15*113` maps to `1234@192.168.15.113`
- `123456*192*168*15*180*5061` maps to `123456@192.168.15.180:5061`

Star Code Script

A star code script is defined with the help of a number of predefined variables and actions. Each variable represents one or one group of configuration parameters. An action can be checking or setting the value of a variable, collecting a phone number from the user, or calling a certain number.

Star Code Script Variables (VAR)

A star code script variable or VAR can be trunk specific or global (non-trunk specific). The general format of a global variable is $var. The general format of a trunk specific variable is TK($var), where TK is the abbreviated name of a trunk (SP1-SP6, BT, or PP). If TK is not specified for a trunk-specific variable, it implies all the applicable trunks in the system.

SP*n* is the SP*n* Service where *n* = 1 through 6 and SP is the OBiTALK Service. Each service is also referred to as a *trunk* in this document.

Here is a list of the supported $var:

- $CFA = call forward unconditional enable (trunk specific; admissible value: 0 for disable, 1 for enable)
- $CFB = call forward busy enable (trunk specific; admissible value: 0 for disable, 1 for enable)
- $CFN = call forward no-answer enable (trunk specific; admissible value: 0 for disable, 1 for enable)
- $CFAN = call forward unconditional number (trunk specific; admissible value: a token representing a call forward number)
- $CFBN = call forward busy number (trunk specific; admissible value: a token representing a call forward number)
- $CFNN = call forward no-answer number (trunk specific; admissible value: a token representing a call forward number)
- $MWS = message waiting state (trunk specific; admissible value: 0 for no new messages, 1 for one or more new messages)
- $DND = do-not-disturb enable (trunk specific; admissible value: 0 for disable, 1 for enable)
- $BAC = block-anonymous caller enable (trunk specific; admissible value: 0 for disable, 1 for enable)
- $BCI = block outbound caller-ID enable (trunk specific; admissible value: 0 for disable, 1 for enable)
- $CWA = call-waiting enable (global; admissible value: 0 for disable, 1 for enable)
- $BCI1 = block caller-ID once (global; admissible value: 1 for enable)
- $UBCI1 = unblock caller-ID once (global; admissible value: 1 for enable)
- $LBM1 = Loopback media (audio samples) once in the next call
- $LBP1 = Loopback RTP packets once in the next call

- $CDM1 = Codecs to enable in the next call (temporarily overriding any codec preferences in device configuration). Each bit of its value represents one audio codec:
  - Bit0 (LSB) = G711u
  - Bit1 = G711a
  - Bit2 = G726r16
  - Bit3 = G726r24
  - Bit4 = G726r32
  - Bit5 = G726r40
  - Bit6 = G729
- $LDN = last dialed number (for redial) (global; read only)
- $BAR1 = Enable Barge-In 1 on the next call (global; admissible value: 1 for enable)
- $Bxrn = Blind Transfer Target Number (global; admissible value: a token representing the target number)
- $LCR = last caller's number (for call return) (global; read only)
- $SPD[n] = number for the speed dial n (n = 1 – 99) (global; admissible value: literal or token representing a phone number)
- $CODE = the digit(s) representing the variable part of a star code (see examples below; read only)

Variable names are case insensitive.

Star Code Script Actions (ACT)

The general format of an action: ACT(par, par, ….)

The following actions are supported:

- set(VAR,token) = Set the given VAR to the value represented by token.
- call(token) = Call the number represented by token. ***Phone Settings***::**OutboundCallRoute** will be applied when making the call (but not the **DigitMap**).
- rpdi(token) = repeat dial the number represented by token.
- coll(VAR) = collect a number from the user and store it as the value of the parameter(s) represented by VAR.The number is collected with ***Phone Settings***::**DigitMap** applied.
- say(token) = display the value represented by token in an on-screen notification message. Values are announced as a list of alphabets or numbers, where token can be a literal (such as 1234) or another variable (such as $CFAN or SP1($CFBN)).
- wifiap = Enable Wi-Fi access point mode for quick Wi-Fi configuration from a web browser.
- blst = Add the number of the last caller to the X_BlockCallers list.

Action names are case insensitive.

Star Code Script Format

General Format: code, name, action1, action2, action3, …

- code = the star code, such as *72. It may contain a variable part enclosed in parentheses, such as *74(x|xx). The variable part as entered by the user are stored in the variable $CODE.
- name = a descriptive name of the function of this star code, such as Call Forward Unconditional.
- action1, action2, … = a valid action with parameters.

Actions are carried out one-by-one in the order as specified in the script.

Restrictions:

- At most 1 coll action per code.
- Either 1 say or 1 call action at most per code, and it must be the last action in the script.

Star Code Script Examples

The following examples are taken from some of the default star code scripts in the OBi device.

- *69, Call Return, call($LCR)

  Calls the number of the caller who rings the PHONE port last time.
- *07, Redial, call($Ldn)

  Redials the last dialed number.
- *72, Call Forward Unconditional, coll($cfan),set($cfa,1)

  Collects a number from the user according to the DigitMap. Then set the CallForwardUnconditionalNumber on all trunks to the collected value, and set the CallForwardUnconditionalEnable on all trunks to Yes.

  To modify the script to enable CallForwardUnconditional on SP1 only, change it to
- *72, Call Forward Unconditional SP1, coll(SP1($cfan)),set(SP1($cfa),1)
- *67, Block Caller ID Once, set($BCI1,1)

  Enable masking of caller ID information once for the next call on any trunk
- *74(x|xx), Set Speed Dial, coll($Spd[$code])

  After user dials *74, OBi expects one or two more digits from the user which represent a speed dial slot index (1 to 99). The 1 or 2-digit variable part is stored in the variable $code.

  OBi device then plays a prompt tone and proceeds to collect a number from the user according to the DigitMap. Finally OBi stores the collected number in the given speed dial slot. If the slot already has a number specified, it will be overwritten quietly with the new value.
- *75(x|xx), Check Speed Dial, say($Spd[$code])

  After user dials *75, OBi expects one or two more digits from the user which represent a speed dial slot index (1 to 99). The 1 or 2-digit variable part is stored in the variable $code.

  The phone displays a message box on the screen to show the number stores in the speed dial slot.

Service Providers

Use the following information for SIP-based configurations. Each ITSP configuration is grouped together as an ITSP profile. The VVX D230 refers to them as ITSP Profile A through ITSP Profile H.

Voice Services:

- SP1–8
- OBiTALK

SIP Service Provider Features

You can configure up to four SIP accounts or SIP Trunks on your system. For the purposes of this guide and elsewhere on the system web interface, documentation, and the OBiTALK portal, the term ITSP describes the logical entity providing the SIP Trunk service to your system. When your system is used with an IP PBX, the IP PBX takes the place of the ITSP if it's the entity providing the SIP Trunk account credential and connectivity to your system.

Each ITSP configuration is grouped together as an ITSP Profile, referred to as ITSP Profiles A through H. The SP service account specifics are grouped under the heading ***SPn Service***, where ***n*** = 1–8. An ITSP Profile includes such parameters as `ProxyServer`, `OutboundProxy`, and `DigitMap`, but doesn't include account-specific parameters. An SP Service includes account-specific parameters such as `AuthUserName` (usually the phone number of the account), `AuthPassword`, `CallerIDName`, and `X_ServProfile` (which ITSP Profile to assume). If the SP Services use the same ITSP, then only one ITSP Profile needs to be configured with all SP Services referred to the same profile.

From your system's point of view, the SP*n* Service using ITSP Profile *X* is enabled with the following minimal settings:

- ***ITSP Profile X – SIP::***`ProxyServer` = *Not Blank*
- ***SPn Service::***`Enabled` = Yes
- ***SPn Service::***`AuthUsername` = *Not Blank*

where ***X*** = A–H and ***n*** = 1–8. Otherwise, the service is considered to be disabled.

SIP Registration

You can configure your systems to register periodically with a SIP Proxy Server or SIP Registration Server. The SIP Proxy Server and SIP Registration Server can be different systems, although in practice they're usually the same. SIP Proxy Server is a required parameter that you must configure on your system. The Registration Server is optional and assumed to be the same as the SIP Proxy Server if it isn't configured on your system.

The main purpose of registration is to create and maintain a dynamic binding of the SIP account to your system's local contact address. The service provider can also rely on this periodic message to infer if your system is online and functional. Each system takes only one local IP address that is either statically assigned in your system's configuration, or dynamically obtained from a local DHCP server. The SP*n* services (for *n* = 1 through 8) each use a different local contact port for sending and receiving SIP messages (defaults are 5060, 5061, 5062, and 5063).

Note that dynamic address binding through periodic registration isn't strictly necessary if the local IP address of your system doesn't change. Your system's contact address can be statically configured on the Registration Server.

SIP Outbound Proxy Server

An outbound proxy server can be configured on your system such that all outbound requests are sent via the outbound proxy server instead of directly to the SIP Proxy Server or Registration Server.

If the outbound proxy server is listening at a nonstandard port, the correct port value must be specified in the `OutboundProxyPort` parameter. The `OutboundProxy` can use a different transport protocol from the `ProxyServer`. The transport protocol to use to communicate with the `OutboundProxy` can be set in the `OutboundProxyTransport` parameters. If `OutboundProxyTransport` is TCP or TLS, your system initiates a TCP or TLS connection only with the `OutboundProxy`. All subsequent messages exchanged between your system and the servers MUST use the same connection. If the connection is closed for any reason, your system attempts to re-establish the connection with the `OutboundProxy` following an exponential back-off retry pattern.

Even though your system only exchanges messages directly with the `OutboundProxy`, the `ProxyServer`, `ProxyServerPort`, and `ProxyServerTransport` parameters are still very much relevant and important, since the SIP requests sent by your system to the server are formed based on these values, not based on the `OutboundProxy` value. The `OutboundProxy` value should never appear in the SIP requests generated by your system, unless the `OutboundProxy` parameter has the same value as `ProxyServer`.

Some server implementations include the outbound proxy server in a Record-Route header such that your system shouldn't respect the locally configured `OutboundProxy` value after the initial INVITE is sent for a new call. This behavior can be achieved by enabling the ***ITSP Profile X – SIP*****::**`X_BypassOutboundProxyInCall` option. However, this option has no effect when the `OutboundProxyTransport` is TCP or TLS, as your system always uses the same connection to send messages to the server.

DNS Lookup of SIP Servers

When sending out SIP requests to the server, your system looks up the IP address of the server using standard DNS query if the server is specified as a domain name instead of an IP address. If an Outbound Proxy Server is configured, it's used instead of the SIP Proxy Server or SIP Registration Server. The resolution of the server domain name into IP address is performed in the following manner:

- Try looking up the name as DNS A Record. If not found,
- Try looking up the name as DNS SRV Record. If not found,
- Try looking up the name as DNS SRV Record with "_sip._udp." prepended to the host name. If not found, fail the request.

If the result from the DNS query is an SRV record, the server port is taken from that record also. The server port value configured on your system is ignored. Otherwise, the server port is taken from the configured value or uses port 5060 if none is specified.

NAT Traversal Considerations

If your system sits behind a NAT router (typically the case), it can discover the mapped external address corresponding to its local SIP contact address as seen by the server in one of the following ways:

- From the “received=” and “rport=” parameters of the VIA header of the REGISTER response sent by the server. These two parameters tell your system its mapped IP address and port number. Your system uses this method if you’ve enabled periodic registration on your system.
- From the response to a STUN binding request your system sent to a STUN server. This method is used by enabling `X_KeepAliveEnable` and setting `X_KeepAliveMsgType` to `stun`. The keep-alive messages are sent to the same server where a REGISTER request would be sent.

Your system always uses the mapped external contact address in all outbound SIP requests instead of its local contact address if one is discovered by either method.

SIP Proxy Server Redundancy and Dual REGISTRATION

Server Redundancy specifically refers to your system’s capability to:

- Look for a working server to REGISTER with from among a list of candidates.
- Change to another server once the server that it currently registers with becomes unresponsive. In other words, system registration must be enabled to use the server redundancy feature.

Other SIP requests, such as INVITE or SUBSCRIBE, are sent to the same server that your system currently registers with.

If Outbound Proxy Server is provided, server redundancy is applied to the Outbound Proxy Server instead of the REGISTRATION server. Server redundancy behavior is enabled by enabling the ***ITSP Profile X – SIP::***`X_ProxyServerRedundancy` parameter, which is disabled by default.

Another requirement for using the server redundancy feature is that the underlying server must be configured in your system as a domain name instead of an IP address. This allows your system to collect a list of candidate servers based on DNS query.

The domain name can be looked up as DNS A record or DNS SRV record. For A records, all the IP addresses returned by the DNS server are considered to have the same priority. For SRV records, the hosts returned by the DNS server can be each assigned a different priority.

After a list of candidate servers is obtained, your system first looks for a working server according to the stated priority. A *working server* means one that your system can successfully register with. This is the *primary server*. Your system maintains registration with the primary server the usual way. However, if no working server is found after traversing the entire list, your system takes a short break and repeats the search in the same order.

While maintaining registration with the primary server, your system continually attempts to fall back to one of the candidate servers that has higher priority than the primary server, if any. The list of candidate servers that your system is trying to fall back on is known as the *primary fallback list*, which may be empty.

In addition, your system can be configured to maintain a secondary registration with a server that has lower or equal priority than the primary server. Secondary registration can be enabled by setting the parameter `X_SecondaryRegistration` to True. If `X_ProxyServerRedundancy` is False, however,

`X_SecondaryRegistration` doesn't take any effect. If this feature is enabled, as soon as a primary server is found, your system searches for a working secondary server in the same manner from the list of candidate servers that are of lower or equal priority than the primary server. In the same way, once your system finds a secondary server, it forms a *secondary fallback list* to continually attempt to fall back on if the list isn't empty.

You can configure the intervals for checking the primary fallback list and the secondary fallback list in the `X_CheckPrimaryFallbackInterval` and `X_CheckSecondaryFallbackInterval` parameters. These parameters are specified in seconds and the default value is `60` for both.

Notes:

- Existence of a secondary server implies a primary server exists.
- If the secondary server exists, it immediately becomes the primary server when the current primary server fails. Your system then starts searching for a new secondary server if the candidate set isn't empty.
- The candidate list can change (be lengthened, shortened, have its priority changed, and so forth) on every DNS renewal (based on the entry's TTL). Your system rearranges the primary and secondary servers and fallback lists accordingly, whichever applies.

If server redundancy is disabled, your system resolves only one IP address from the server's domain name, and won't try other IP addresses if the server isn't responding.

SIP Privacy

Your system observes inbound caller privacy and decodes the caller's name and number from SIP INVITE requests by checking the FROM, P-Asserted-Identity (PAID), and Remote-Party-ID (RPID) message headers. All these headers can carry the caller's name and number information.

If PAID is present, system takes the name and number from it. Otherwise, it takes the name and number from RPID if present, or from the FROM header otherwise. RPID, if present, includes the privacy setting desired by the caller. This privacy can indicate one of the following options:

- *off* = no privacy requested. Your system shows name and number.
- *full* = full privacy requested. Your system hides both name and number.
- *name* = name privacy requested. Your system shows the number but hides the name.
- *uri* = uri privacy requested. Your system shows the name but hides the number.

Regardless, if PAID exists or not, your system always takes the privacy setting from the RPID if it's present in the INVITE request. Note that if the resulting caller name is "Anonymous" (case-insensitive), system treats it as if the caller is requesting full privacy.

For outbound calls, the caller's preferred privacy setting can be stated by your system in an RPID header of the outbound INVITE request. To enable this behavior, set the ***ITSP Profile X – SIP***::`X_InsertRemotePartyID` parameter to True, which is the default value of this parameter. Your system supports only two outbound caller privacy settings: privacy=off or privacy=full. The RPID header generated by your system carries the same name and number as the FROM header. If outbound caller-ID is blocked, your system sets privacy=full in RPID, and also sets the display name in the FROM and RPID headers to "Anonymous" for backward compatibility. Your system won't insert PAID in outbound INVITE requests.

STUN and ICE

Your system supports standard STUN based on RFC3489 and RFC5389 for passing inbound RTP packets to systems sitting behind NATs. The parameters that control the STUN feature are found in the **ITSP Profile X – General::** section:

- `STUNEnable` – Enables this feature. Default is False.
- `STUNServer` – The IP address or domain name of the external STUN server to use. STUN feature is disabled if this value is blank, which is the default.
- `X_STUNServerPort` – The STUN Server's listening UDP port. Default value is `3478` (standard STUN port).

The STUN feature used in this context is only for RTP packets, not SIP signaling packets, which typically don't require STUN. Your system sends a STUN binding request right before making or answering a call on SP1/2. If the request is successful, your system decodes the mapped external address and port from the binding response and uses them in the `m=` and `c=` lines of its SDP offer or answer sent to the peer system. If the request fails, such as STUN server not found or not responding, the call goes on without using external address in the SDP.

Standard RTP requires using an even-numbered port in the `m=` line. If the external port isn't an even number, your system changes the local RTP port and redoes STUN, and continues to do this as many as four times or until an even external port number is found. If the fourth trial still results in an odd external port number, the call goes on without using an external address in the SDP.

Your system supports standard ICE based on RFC5245. ICE is done on a per-call basis for automatically discovering which peer address is the best route for sending RTP packets. To enable ICE on your system, set the ***ITSP Profile X – General*****::**`X_ICEEnable` parameter to True. The default is False.

ICE is effective if STUN is also enabled. However, STUN not a requirement for using ICE on your system. If STUN is enabled and an external RTP address different from its local address is discovered, your system offers two ICE candidates in its SDP:

- The local (host) address (highest priority)
- The external (srflx or server reflexive) address

Otherwise, only the local host candidate is shown in your system's SDP. Note that your system uses the srflx address in the `m=` and `c=` lines of the SDP if STUN is enabled and successful.

If ICE is enabled and the peer's SDP has more than one candidate, your system sends STUN requests to each peer candidate from its local RTP port. As soon as it receives a response from the highest priority candidate, your system concludes ICE and uses this candidate to communicate with the peer. Otherwise, your system allows as long as 5 seconds to wait for the response from all the candidates, and selects the highest priority candidate that has a response. Once ICE completes successfully, your system further applies symmetric RTP concept to determine the peer's RTP address (that is, sends them to the address from which the peer's RTP packets are coming).

RTP Statistics – the X-RTP-Stat Header

When ending an established call, your system can include a summary of the RTP statistics collected during the call in the SIP BYE request or the 200 response to the SIP BYE request sent by the peer system. The summary is carried in an X-RTP-Stat header in the form of a comma-separated list of fields. The reported fields are:

- `PS` = Number of Packets Sent

- PR = Number of Packets Received
- OS = Number of bytes sent
- OR = Number of bytes received
- PL = Number of packets lost
- JI = Jitter in milliseconds
- LA = Decode latency or jitter buffer size in milliseconds
- DU = Call duration in seconds
- EN = Last Encoder Used
- DE = Last Decoder Used

For example:

```
X-RTP-Stat:PS=1234,OS=34560,PR=1236,OR=24720,JI=1,DU=1230,PL=0,EN=G711U, DE=G711U
```

To enable the X-RTP-Stat feature, set the ***ITSP Profile X – SIP*****::**X_InsertRTPStats parameter to True.

Media Loopback Service

Your system supports the media loopback draft as described in *draft-mmusic-media-loopback-13.txt*. You can enable or disable this feature from ***System Management – Device Admin – Media Loopback***.

Your system supports the following media loopback features:

- Loopback modes: loopback-source and loopback-mirror
- Loopback types: rtp-media-loopback and rtp-packet-loopback
- Loopback packet formats: encaprtp, loopbkprimer

When your system acts as a loopback mirror, it always sends primer packets so that incoming packets can get through NAT/Firewall. The following parameters (in the ***System Management – Device Admin – Media Loopback*** section) control the media loopback feature:

- AcceptMediaLoopback – Enable system to accept an incoming call that requests media loopback. Default is True.
- MediaLoopbackAnswerDelay – The delay in ms before your system answers a media loopback call. Default is 0.
- MediaLoopbackMaxDuration – The maximum duration to allow for an incoming media loopback call. Default is 0, which means the duration is unlimited.

Your system rejects an incoming media loopback call if:

- Handset port is off-hook.
- Handset port is ringing.

Your system terminates an inbound media loopback call already in progress when:

- Handset port is off-hook.
- Handset port is ringing.

To make an outgoing loopback call, dial one of the following star codes before dialing the target number:

- *03 – Make a Media loopback call.
- *04 – Make an RTP packet loopback call.

Note that an outbound Media Loopback Call isn't subject to a call duration limit. It lasts until you hang up or until the called system ends the call.

For more information on ITSP parameters, see the VVX D230 Data Model.

Use SPn as a Proxy for a SIP IP Phone

An SP service can be set up as a proxy for a legacy IP phone to let the phone access OBiTALK on your system. This proxy mode of operation must be explicitly enabled in the SP 's configuration on your system. It's disabled by default. The IP phone using this proxy service is the *local_client* of the SP service. It must be installed on the LAN side of your system.

In this mode, SP*n* accepts SIP Registration from the client system from the LAN side, which must use the same user-id and password as this SP*n*'s `AuthUserName` and `AuthPassword` parameters for authentication. This client system can also send SIP INVITE to your system at this SP to make calls. This SP's `InboundCallRoute` must be set up with the proper routing rule to handle calls from the *local_client*.

Send the SIP Proxy Server parameter on the client system to:

```
<obi-number>.pnn.obihai.com:<spn-user-agent-port>
```

where `<obi-number>` is the 9-digit OBi number of this system, and `<spn-user-agent-port>` is SP*n*'s `X_UserAgentPort` parameter.

For example, SP1 has a *local_client* with the user-id `4086578118`. The client wishes to make and receive calls on SP3. The SP1 **InboundCallRoute** shall include the following rule:

```
{4086578118>:sp3}
```

The SP3 `InboundCallRoute` shall be: `{sp1(`**`408657118@local_client`**`)}`

For more information on SP*n* services parameters and calling feature parameters, see the VVX D230 Data Model.