UNIFY Harmonize your enterprise

OpenScape Office

WSI interface

We are now Unify.

This document references our previous company name; all othercontent is still valid and correct. For further information, please visit unify.com

UNIFY Harmonize your enterprise

OpenScape Office V3

Description

Web Services Interface

Version 1.0

History of Change

Version	Date	Comments
0.5	2011-02-28	First Draft
1.0	2011-10-07	Minor corrections

Content

1 General usage

The Web Services Interface (WSI) Reference will help you to use the Application Programming Interface (API) of the Web Services. All provided functions are grouped in the following categories

- Call control
- Device control
- Monitoring
- EVM Entry Voicemail (HiPath 3000 / Entry Web Services only)

Within the categories, the functions are sorted in alphabetical order.

Every section consists of a description of the parameters, an https-request example, a short description how the example works and tables containing the response messages that are sent by the Web Services after a request was received and/or has been fulfilled.

In order to send any command, the client program or web page has to be successfully authenticated towards the web Services via login and password. It is necessary that the logged-in user has sufficient access rights to control certain devices.

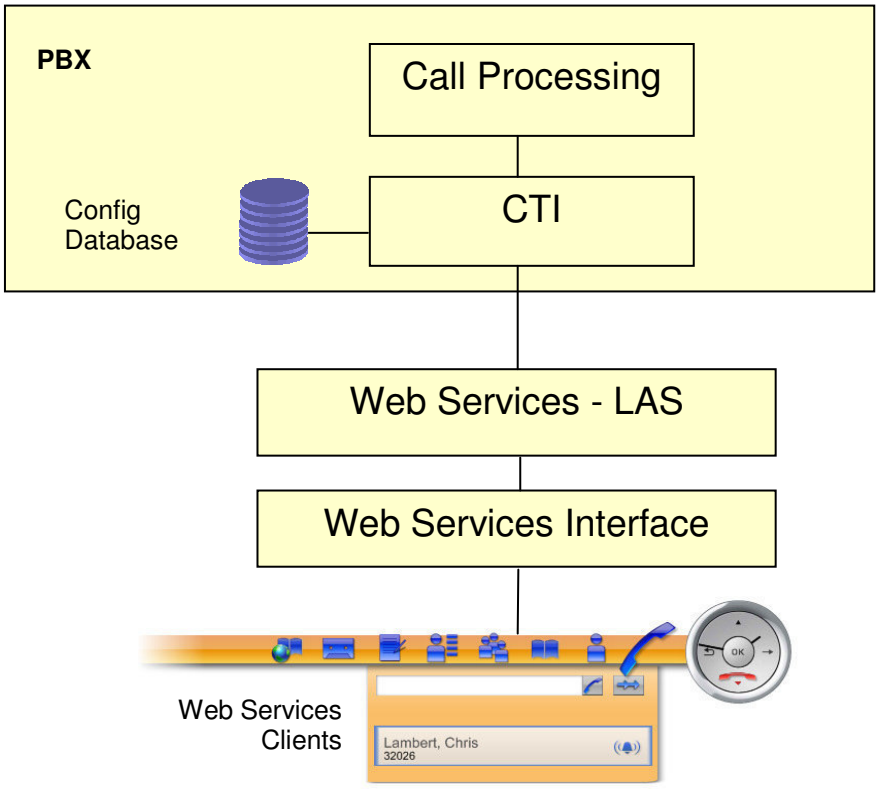

Figure 1 Schema of the interworking between all components

1.1 CGI Syntax

The Web Services Interface (WSI) is accessible via https- or http-requests with CGI syntax. Requests send to the Web Services are composed of an address part and a command part.

Address part

http://[WebServices-IP-address]:8801/cgi-bin/
or
https://[WebServices-IP-address]:8802/cgi-bin/

Command part

gadgetapi?cmd=[CommandName]&[ParameterName]=[ParameterValue]&[ParameterName]=[Param eterValue]&...

Simple example in pure html environment

```
<html>
<body>
<button onclick="request('http://[Web Services IP address]:8801/cgi-
bin/gadgetapi?cmd=MakeCall&callingDevice=103&calledDirectoryNumber=101 ')">
Send Request
</button>
</body>
</html>
```

1.2 Parameter Types

Parameter	Type	Syntax	Description
<name>	**command**	ANSI text string	The command to call a certain function.
<name>	**deviceID**	internal number use of [0-9]	The device which will be controlled. Device ID refers always to the configured phone number of the appropriate device.
<name>	**Boolean**	“true” or “false”	In order to enable (true) or disable (false) a function.
<name>	**eventtype**	JSON String	Used for filter parameters
<name>	**snapshotObject**	CSTA XML	Special type used in order to snapshot a device.

Table 1 The CGI parameter values can have a different type and syntax.

1.3 Call Number handling

All phone or extension numbers used in the Web Services depends on their field of usage. The numbers have to be entered with in following format:

- Internal number: <XX>or<XXX> or <XXXX> or <XXXXX>
- External number: +<country code> (<regional code>) <target number>

Example:

- Internal number: 101
- External number: +49 (89) 123456789

It's strongly recommended to use only the canonical format for any external phone numbers.

1.4 Devices

All commands, that require control of a phone, like the "MakeCall" command, can only be executed when the phone, initiating the action is an HFA phone. No SIP phone can be controlled in a way that it starts with an action. For this reason phone clients with HFA capabilities are preferred.
All described functions in this document have been tested with IP-phones of the type OptiPoint and OpenStage using HFA software, if not otherwise stated. All these phones are business phones, that provide features like hands-free-mode, forwarding, initiate meetings, etc.

Device list with full functionality:

- OptiPoint 410 standard HFA v5.1 or higher
- OptiPoint 410 advanced HFA v5.1 or higher
- OptiPoint 420 standard HFA v5.1 or higher
- OpenStage HFA v1R2.5.0 or higher

1.5 Preconditions for using the examples

At every section you can find simple examples.

If you want to execute the html request the following circumstances had to be fulfilled in order to let the examples work:

- Be sure using the right protocol http (Port 8801) or https (Port 8802).
- Be sure using your own network and device configuration (IP-address and device numbers).
- For this simple example a valid login towards the Web Services is absolute necessary, before using the examples.
- For controlling a certain device, it is absolute necessary that the logged in user has the rights to control this device.

1.6 List of possible requests, their response type and examples

Request	Response Type	Response
Command : MakeCall Parameter : callingDevice,calledDirectoryNumber	XML	<?xml version="1.0" encoding="UTF-8"?> <MakeCallResponse> <callingDevice> <deviceID>114</deviceID> <callID>00000010</callID> </callingDevice> </MakeCallResponse>
Command : ClearConnection Parameter : deviceID	XML	<?xml version="1.0" encoding="UTF-8"?> <ClearConnectionResponse/>
Command : AnswerCall Parameter : deviceID	XML	<?xml version="1.0" encoding="UTF-8"?> <AnswerCallResponse/>
Command : HoldCall Parameter : deviceID	XML	<?xml version="1.0" encoding="UTF-8"?> <HoldCallResponse/>
Command : RetrieveCall Parameter : deviceID	XML	<?xml version="1.0" encoding="UTF-8"?> <RetrieveCallResponse/>
Command : DeflectCall Parameter : deviceID,newDestination	XML	<?xml version="1.0" encoding="UTF-8"?> <DeflectCallResponse/>
Command : SingleStepTransferCall Parameter : deviceID,transferredTo	XML	<?xml version="1.0" encoding="UTF-8"?> <SingleStepTransferCallResponse> <transferredCall> <deviceID>118</deviceID> <callID>0017</callID> </transferredCall> </SingleStepTransferCallResponse>
Command : SetDisplay Parameter : device,contentsOfDisplay	XML	<?xml version="1.0" encoding="UTF-8"?> <SetDisplayResponse/>

Command : ClearDisplay Parameter : device	XML	<?xml version="1.0" encoding="UTF-8"?> <SetDisplayResponse/>
Command : SnapshotDevice[1] Parameter : snapshotObject	XML	<?xml version="1.0" encoding="UTF-8"?> <SnapshotDeviceResponse> <crossRefIDorSnapshotData> <snapshotData> <snapshotDeviceResponseInfo> <connectionIdentifier> <deviceID>116</deviceID> <callID/> </connectionIdentifier> <localCallState> <compoundCallState> <localConnectionState>null</localConnectionState> </compoundCallState> </localCallState> </snapshotDeviceResponseInfo> </snapshotData> </crossRefIDorSnapshotData> </SnapshotDeviceResponse>
Command : GetForwarding Parameter : device	XML	<?xml version="1.0" encoding="UTF-8"?> <GetForwardingResponse> <forwardingList> <forwardListItem> <forwardingType>forwardImmediate</forwardingType> <forwardStatus>false</forwardStatus> </forwardListItem> </forwardingList> </GetForwardingResponse>
Command : SetForwarding Parameter : device,forwardDN,activateForward	XML	<?xml version="1.0" encoding="UTF-8"?> <SetForwardingResponse/>
Command : GetDoNotDisturb Parameter : device	XML	<?xml version="1.0" encoding="UTF-8"?> <GetDoNotDisturbResponse> <doNotDisturbOn>false</doNotDisturbOn> </GetDoNotDisturbResponse>
Command : SetDoNotDisturb Paramater : device,doNotDisturbOn	XML	<?xml version="1.0" encoding="UTF-8"?> <SetDoNotDisturbResponse/>
Command : MonitorStart Parameter : deviceObject	JSON	{"crossRefID":"0001","error":0,"respType":"MonitorStartResponse","MonitorCount":3}
Command : MonitorStop Parameter : deviceObject	JSON	{"crossRefID":"0001","error":0,"respType":"MonitorStopResponse","MonitorCount":2}
Command : GetEvents Parameter : deviceObject	JSON	{"error":0,"events":[]}
Command : GetInstantEvents Parameter : deviceObject	JSON	{"error":0,"events":[]}

Table 2 Overview of possible requests with response type and an example for a valid response

[1] This part of the Web Services Interface is deprecated. It is not recommended to use it.

2 Login and Logout

2.1 Login

Description

The login is mandatory to authenticate the user.
Format:gsUser=100&gsPass=1234

Parameter	Type	Description
gsUser	**deviceID**	the user ID
gsPass	**deviceID**	the user's password.

Return Values

positive

```
<LOGIN>
<ID>sessionID</ID>
<CNT>count</CNT>
</LOGIN>
```

where sessionID identifies the session for further requests. It can be ignored because the server also stores it on the client side with a cookie. The "count" shows the number of sessions for a user.

negative

If the login fails:

```
<LOGIN>
<ID>0</ID>
<ERROR>reson</ERROR>
</LOGIN>
```

When the login fails the ID is always zero and the <ERROR> tag is present the reason can have the following values:

- LOGIN_FAILED: Password wrong or user unknown, user not allowed to login
- NOKEY: No license key.

2.2 Logout

Description

Ends the current session. The session is identified with the stored cookie.
Format: cmd=Logout

Response

<LOGOUT>

3 Call control

3.1 AnswerCall

Description

The AnswerCall service connects an alerting or queued call. The call will be answered in hands-free mode of the phone.

Parameter	Type	Description
alertingDevice	**deviceID**	The device which is ringing and should be answering the call.

Return Values

positive

Error Code	Response Type
0	AnswerCallResponse

negative

Error Code	Response Type	Error Response
1	CSTAErrorCode	invalidObjectState

Example https-request

http://[Web Services IP address]:8801/cgi-bin/gadgetapi?cmd=AnswerCall&deviceID=103

Example https-request Description

The device 103 is answering the call in hands-free mode.

Example possible, positive XML response from Web Services

```
<?xml version="1.0" encoding="UTF-8"?>
<AnswerCallResponse/>
```

3.2 ClearConnection

Description

Clear connection terminates the current call for a specific phone. The call is finished if the calling device or the called device is receiving this command.

Parameter	Type	Description
deviceID	**deviceID**	A device participating in a call that should be cleared.

Return Values

positive

Error Code	Response Type
0	ClearConnectionResponse

negative

Error Code	Response Type	Error Response
1	CSTAErrorCode	invalidCallID
1	CSTAErrorCode	invalidObjectState

Example https-request

http://[Web Services IP address]:8801/cgi-bin/gadgetapi?cmd=AnswerCall&deviceID=103

Example https-request Description

The connection of an active call of the phone 103 will be interrupted.

Example possible, positive XML response from Web Services

```
<?xml version="1.0" encoding="UTF-8"?>
<ClearConnectionResponse/>
```

3.3 DeflectCall

Description
This function will transfer an incoming call from the original target to a new target. The first target will be not affected by this action.

Parameter	Type	Description
deviceID	deviceID	The device ID of the alerting phone.
newDestination	deviceID	The device ID for the target of the deflection.

Return Values

positive

Error Code	Response Type
0	DeflectCallResponse

negative

Error Code	Error Response	Response Type
1	CSTAErrorCode	invalidCallID
1	CSTAErrorCode	resourceBusy
1	CSTAErrorCode	invalidDeviceID

Example https-request

http://[Web Services IP address]:8801/cgi-bin/gadgetapi?cmd=DeflectCall&deviceID=103&newDestination=109

Example https-request Description

Transfers a call, without answering from device 103 to device 109. Device 103 is ringing and the call should move to another phone without conversation. The DeflectCall command moves the call from the ringing 103 to 109. Phone 109 is ringing now and phone 103 is put back into normal mode with no activity.

Example possible, positive XML response from Web Services

```
<?xml version="1.0" encoding="UTF-8"?>
<DeflectCallResponse/>
```

3.4 HoldCall

Description
Puts an active call on hold. Can only be used if there is an active and answered call.

Parameter	Type	Description
deviceID	**deviceID**	The deviceID of the phone number which initiates the Hold

Return Values

positive

Error Code	Response Type
0	HoldCallResponse

negative

Error Code	Response Type	Error Response
1	CSTAErrorCode	atLeastOneConditionalParameterNotProvided

Example https-request

http://[Web Services IP address]:8801/cgi-bin/gadgetapi?cmd=HoldCall&deviceID=103

Example https-request Description

If device 103 is in an active call, the command puts the active call on hold for other activities.

Example possible, positive XML response from Web Services

```
<?xml version="1.0" encoding="UTF-8"?>
<HoldCallResponse/>
```

3.5 MakeCall

Description
Initiates a basic call between the calling device and target device (called device), addressed by the calledDirectoryNumber. This is possible when no active call is established and the handset is placed on the phone. In order to establish a call the source phone will active the hands free mode and initiates the call towards the target.

Parameter	Type	Description
callingDevice	**deviceID**	The device which should be performing the call.
calledDirectoryNumber	**deviceID**	The target device that should be called.

Return Values

positive

Error Code	Response Type
0	MakeCallResponse

negative

Error Code	Response Type	Error Response
1	CSTAErrorCode	generic
1	CSTAErrorCode	invalidCalledDeviceID

Example https-request

http://[Web Services IP address]:8801/cgi-bin/gadgetapi?cmd=MakeCall&callingDevice=101&calledDirectoryNumber=103

Example https-request Description

The device 101 initiates a call to device 103. The device 101 is in hands-free-mode and the device 103 is ringing.

Example possible, positive XML response from Web Services

```
<?xml version="1.0" encoding="UTF-8"?>
<MakeCallResponse>
  <callingDevice>
    <deviceID>103</deviceID>
    <callID>00000010</callID>
  </callingDevice>
</MakeCallResponse>
```

3.6 RetrieveCall

Description
Gets back a hold call. This can be used to end the consultation call and get back to the original (held) call. It is only possible for calls that are in hold mode.

Parameter	Type	Description
deviceID	**deviceID**	Device that wants to get back a held call.

Return Values

positive

Error Code	Response Type
0	RetrieveCallResponse

negative

Error Code	Response Type	Error Response
1	CSTAErrorCode	atLeastOneConditionalParameterNotProvided

Example https-request

http://[Web Services IP address]:8801/cgi-bin/gadgetapi?cmd=RetrieveCall&deviceID=103

Example https-request Description

The held call of device 103 is reactivated. The held call is active for conversation now.

Example possible, positive XML response from Web Services

```
<?xml version="1.0" encoding="UTF-8"?>
<RetrieveCallResponse/>
```

3.7 TransferCall

Description
This request is used to make a blind transfer, without making a previous consultation to the target device. This feature is also known under the name “Single Step Transfer”. Condition for this is that a call is active between you phone and the calling device and the target phone has no active call running.

Parameter	Type	Description
deviceID	**deviceID**	The calling device.
transferredTo	**deviceID**	The target device.

Return Values

positive

Error Code	Response Type
0	SingleStepTransferCallResponse

negative

Error Code	Response Type
1	atLeastOneConditionalParameterNotProvided

Example https-request

http://[Web Services IP address]:8801/cgi-bin/gadgetapi?cmd=SingleStepTransferCall&deviceID=103&transferredTo=108

Example https-request Description

For example there is a call established between the devices 100 and 103. The https-request transfers this call for speaking between device 103 and device 108. The target is that devices 103 and 108 can make conversation.

Example possible, positive XML response from Web Services

```
<?xml version="1.0" encoding="UTF-8"?>
<SingleStepTransferCallResponse>
  <transferredCall>
    <deviceID>103</deviceID>
    <callID>0017</callID>
  </transferredCall>
</SingleStepTransferCallResponse>
```

4 Device control

4.1 ClearDisplay

Description
Clears any text set by SetDisplay from a certain phone. The phones standard display will appear again. If nothing is displayed on the phone this command won´t have any effect. This will work together with OptiPoint HFA and OpenStage HFA phones.

Parameter	Type	Description
Device	**deviceID**	The target phone.

Return Values

positive

Error Code	Response Type
0	SetDisplayResponse

negative

Error Code	Response Type	Error Response
1	CSTAErrorCode	invalidCallID
1	CSTAErrorCode	resourceBusy
1	CSTAErrorCode	invalidDeviceID

Example https-request

http://[Web Services IP address]:8801/cgi-bin/gadgetapi?cmd=ClearDisplay&device=103

Example https-request Description

Clears the display of device 103.

Example possible, positive XML response from Web Services

```
<?xml version="1.0" encoding="UTF-8"?>
<SetDisplayResponse/>
```

4.2 GetDoNotDisturb

Description
Requests the “do-not-disturb” status for a certain phone. For incoming call the phone is not available as valid target.

Parameter	Type	Description
Device	**deviceID**	The phone which “DoNotDisturb” status should be read.

Return Values

positive

Error Code	Response Type
0	True
0	False

negative

Error Code	Response Type	Error Response
1	CSTAErrorCode	invalidObjectState

Example https-request

http://[Web Services IP address]:8801/cgi-bin/gadgetapi?cmd=GetDoNotDisturb&device=103

Example https-request Description

The “do-not-disturb” status for device 103 is not (false) activated. This information can be found in the XML answer of this request.

Example possible, positive XML response from Web Services

```
<?xml version="1.0" encoding="UTF-8"?>
<GetDoNotDisturbResponse>
  <doNotDisturbOn>false</doNotDisturbOn>
</GetDoNotDisturbResponse>
```

4.3 GetForwarding

Description
Retrieves forwarding configuration for target device.

Parameter	Type	Description
device	**deviceID**	The phone that's configuration should be read.

Return Values

positive

Error Code	Response Type
0	GetForwardingResponse

negative

Error Code	Response Type	Error Response
1	CSTAErrorCode	invalidObjectState

Example https-request

http://[Web Services IP address]:8801/cgi-bin/gadgetapi?cmd=GetForwarding&device=103

Example https-request Description

Gets the forwarding configuration status for the device 103.

Example possible, positive XML response from Web Services

```
<?xml version="1.0" encoding="UTF-8"?>
<GetForwardingResponse>
  <forwardingList>
    <forwardListItem>
      <forwardingType>forwardImmediate</forwardingType>
      <forwardStatus>false</forwardStatus>
    </forwardListItem>
  </forwardingList>
</GetForwardingResponse>
```

4.4 SetDisplay

Description
Set the display text for a certain phone. In case of no other application is overwriting this space, the text will not be deleted after time and has to be deleted actively by the application.

Parameter	Type	Description
device	**deviceID**	The target phone.
contentsOfDisplay	**Text**	The display text.

Return Values

positive

Error Code	Response Type
0	SetDisplayResponse

negative

Error Code	Response Type	Error Response
1	CSTAErrorCode	invalidCallID
1	CSTAErrorCode	resourceBusy
1	CSTAErrorCode	invalidDeviceID

Example https-request

http://[Web Services IP address]:8801/cgi-bin/gadgetapi?cmd=SetDisplay&device=103&contentsOfDisplay=testtext

Example https-request Description

Writes the text “testtext” onto the display of device 103.

Example possible, positive XML response from Web Services

```
<?xml version="1.0" encoding="UTF-8"?>
<SetDisplayResponse/>
```

4.5 SetDoNotDisturb

Description
Set “do-not-disturb status” for a certain phone. If this feature is activated every call will be blocked. No forward is activated by this function.

Parameter	Type	Description
device	**deviceID**	The phone that should be set.
doNotDisturbOn	**Boolean**	Enable / Disable the setting with this parameter.

Return Values

positive

Error Code	Response Type
0	SetDoNotDisturbResponse

negative

Error Code	Response Type	Error Response
1	CSTAErrorCode	invalidObjectState
1	CSTAErrorCode	invalidDeviceID

Example https-request

http://[Web Services IP address]:8801/cgi-bin/gadgetapi?cmd=SetDoNotDisturb&device=103&doNotDisturbOn=true
http://[Web Services IP address]:8801/cgi-bin/gadgetapi?cmd=SetDoNotDisturb&device=103&doNotDisturbOn=false

Example http-request Description

The first example activates the “do-not-disturb” status for device 103. The second one disables the “do-not-disturb” status for device 103.

Example possible, positive XML response from Web Services

```
<?xml version="1.0" encoding="UTF-8"?>
<SetDoNotDisturbResponse/>
```

4.6 SetForwarding

Description
A certain phone is ordered to forward all incoming calls to another device (unconditioned forwarding). This feature can be deactivated too. If a forward is active, a second forward with “activeforward=true” parameter will replace the first one.

Parameter	Type	Description
device	**deviceID**	The phone that should forward incoming calls.
forwardDN	**deviceID**	The phone that should receive the calls instead.
activateForward	**Boolean**	Enable / Disable the setting with this param.

Return Values

positive

Error Code	Response Type
0	SetForwardingResponse

negative

Error Code	Response Type	Error Response
1	CSTAErrorCode	invalidForwardingDestination
1	CSTAErrorCode	invalidObjectState
1	CSTAErrorCode	operation:generic

Example http-request

http://[Web Services IP address]:8801/cgi-bin/gadgetapi?cmd=SetForwarding&device=103&forwardDN=109&activateForward=true
http://[Web Services IP address]:8801/cgi-bin/gadgetapi?cmd=SetForwarding&device=103&forwardDN=109&activateForward=false

Example http-request Description

In the first example the forward destination is programmed. Now every call for device 103 is send to device 109. The forwarding target is shown on the display of the phone.
The second example is to delete the forward destination. All calls will arrive at device 103 again by sending this command.

Example possible, positive XML response from Web Services

```
<?xml version="1.0" encoding="UTF-8"?>
<SetForwardingResponse/>
```

4.7 SnapshotDevice

Description

This function is monitoring the call_ID of a call, if the device is calling. For some features, it is necessary to know the call_ID. The call_ID is a string with numbers and letters that is unique for every active call running at the moment. The call_ID normally it was counted in hex format. The call_ID is only available as long as a call is active. After the end the call_ID is freed for later use. There will be only an answer of the call ID, when the object has an active call running. The call_ID has nothing to do with the phone number. The relation between the phone numbers and a call_ID is only given as long as the connection between both phones are active.

Parameter	Type	Description
snapshotObject	**snapshotObject**	In order to get the Call ID of an active call, one device that is involved in this active call has to be filled in.

Return Values
positive

Error Code	callID	Response Type
0	000000a6	SnapshotDeviceResponse

The callID is only an example.

negative

Error Code	Response Type	Error Response
1	CSTAErrorCode	invalidDeviceID

Example http-request

http://[Web Services IP address]:8801/cgi-bin/gadgetapi?cmd=SnapshotDevice&snapshotObject=103

Example http-request Description

This command delivers the call ID of an active call of device 103.

Example possible, positive XML response from Web Services

```
<?xml version="1.0" encoding="UTF-8"?>
<SnapshotDeviceResponse>
  <crossRefIDorSnapshotData>
    <snapshotData>
      <snapshotDeviceResponseInfo>
        <connectionIdentifier>
          <deviceID>116</deviceID>
          <callID/>
        </connectionIdentifier>
        <localCallState>
          <compoundCallState>
            <localConnectionState>null</localConnectionState>
          </compoundCallState>
        </localCallState>
      </snapshotDeviceResponseInfo>
    </snapshotData>
  </crossRefIDorSnapshotData>
</SnapshotDeviceResponse>
```

5 Monitoring

5.1 GetEvents

Description
Fetches all events for the given device object. The function returns events only if the device has an active monitor and events occur. If no events can be obtained immediately, the Web Services holds the connection for 5 seconds and responses immediately if events occur in this period of time. Of course you can query the Web Services again if there were no events. If you want to view the events permantently, it is required to reactivate the GetEvents function after every event and every 5 seconds. It is absolute necessary to use MonitorStart before, otherwise the GetEvents function will not receive any events.
This is often used for detecting incoming calls. If an event occurs in the time period of five seconds it is absolute necessary to restart the function, in order to notice the next event.

Parameter	Type	Description
deviceObject	**deviceID**	The phone where monitoring should start stop.
Filter	**eventtype**	Optional. Get only events matching the filter.

Event types supported by the filter:

- ServiceInitiatedEvent
- OriginatedEvent
- DeliveredEvent
- ConnectionClearedEvent
- EstablishedEvent
- FailedEvent
- TransferedEvent

Return Values Example:

positive

"error":0,"events":[{"calledDevice":"100","deviceID":"100","type":"DeliveredEvent","callID":"00000232","callingDevice":"102"}]} on success.

If there is no event include in the 5 seconds there will be an answer like this with no events information:

```
{"error":0,"events":[]}
```

The result is of the type “JSON string”, containing an array of events or an empty array. If there were no events the array is empty and in that case just poll again. The client application has to evaluate this JSON string.

negative

Error Code	Response Type
1	deviceHasNoMonitor(no Monitor started)

Example http-request

http://[Web Services IP address]:8801/cgi-bin/gadgetapi?cmd=GetEvents&deviceObject=103

Example http-request Description

Gets the events from PBX for device 103.

Possible events

The events returned by the GetEvents service of the Web Services have a JSON structure. The returned value can contain zero or more events. Hence the single events are packed in an Array with syntax: [{...},{...},...,{...}] A returned value with no events would look like this: [].

All events have a type, EventForDevice and a receivedAt element in their content. The type of an element defines the structure of its content. Different event types have different additional content and a jsonEvent where all event types are shown that are responded by the PBX. In the following two tables the event type, the parameters and a jsonEvent example are listed.

Event type	*Parameters*
ServiceInitiatedEvent	deviceID, callID, cause, initiatingDevice
OriginatedEvent	deviceID, callID, callingDevice, calledDevice
DeliveredEvent	deviceID, callID, callingDevice, calledDevice, cause
ConnectionClearedEvent	deviceID, callID, callerID*, calledID*
EstablishedEvent	establishedConnection, answeringDevice, callingDevice
RetrievedEvent	deviceID, retrievingDevice, callID
HeldEvent	deviceID, holdingDevice, callID
DivertedEvent	deviceID, newDestination, callID
DoNotDisturbEvent	device
ForwardingEvent	deviceID, status
FailedEvent	
TransferedEvent	transferringDevice, transferredToDevice

(*)optional

Event type	*Parameters & example for jsonEvent*
ServiceInitiatedEvent	deviceID, callID, cause, initiatingDevice `"jsonEvent":{"ServiceInitiatedEvent":{"localConnectionInfo":"","monitorCrossRefID":"","cause":"","initiatedConnection":{"deviceID":"","callID":""},"initiatingDevice":{"deviceIdentifier":""}}}`
OriginatedEvent	deviceID, callID, callingDevice, calledDevice `"jsonEvent":{"OriginatedEvent":{"calledDevice":{"deviceIdentifier":""},"localConnectionInfo":"","monitorCrossRefID":"","cause":"","originatedConnection":{"deviceID":"","callID":""},"callingDevice":{"deviceIdentifier":""}}}`
DeliveredEvent	deviceID, callID, callingDevice, calledDevice, cause `"jsonEvent":{"DeliveredEvent":{"calledDevice":{"deviceIdentifier":""},"localConnectionInfo":"","connection":{"deviceID":"","callID":""},"monitorCrossRefID":"","cause":"","alertingDevice":{"deviceIdentifier":""},"callingDevice":{"deviceIdentifier":""}}}`

ConnectionClearedEvent	deviceID, callID, callerID*, calledID* `"jsonEvent":{"ConnectionClearedEvent":{"localConnectionInfo":"","monitorCrossRefID":"","cause":"","releasingDevice":{"deviceIdentifier":""},"droppedConnection":{"deviceID":"","callID":""}}}`
EstablishedEvent	establishedConnection, answeringDevice, callingDevice `"jsonEvent":{"EstablishedEvent":{"calledDevice":{"deviceIdentifier":""},"answeringDevice":{"deviceIdentifier":""},"localConnectionInfo":"","monitorCrossRefID":"","cause":"","establishedConnection":{"deviceID":"","callID":""},"callingDevice":{"deviceIdentifier":""}}}`
RetrievedEvent	deviceID, retrievingDevice, callID `"jsonEvent":{"RetrievedEvent":{"localConnectionInfo":"","monitorCrossRefID":"","cause":"","retrievingDevice":{"deviceIdentifier":""},"retrievedConnection":{"deviceID":"","callID":""}}}`
HeldEvent	deviceID, holdingDevice, callID `"jsonEvent":{"HeldEvent":{"localConnectionInfo":"","monitorCrossRefID":"","cause":"","heldConnection":{"deviceID":"","callID":""},"holdingDevice":{"deviceIdentifier":""}}}`
DivertedEvent	deviceID, newDestination, callID
DoNotDisturbEvent	device `"jsonEvent":{"DoNotDisturbEvent":{"monitorCrossRefID":"","doNotDisturbOn":"","device":""}}`
ForwardingEvent	deviceID, status `"jsonEvent":{"ForwardingEvent":{"monitorCrossRefID":"","forwardingType":"forwardingImmediate","device":"","forwardTo":"","forwardStatus":""}}`
FailedEvent	`"jsonEvent":{"FailedEvent":{"calledDevice":{"deviceIdentifier":"116"},"localConnectionInfo":"connected","monitorCrossRefID":"0000","cause":"doNotDisturb","failedConnection":{"deviceID":"116","callID":"0000000a"},"callingDevice":{"deviceIdentifier":"114"},"failingDevice":{"deviceIdentifier":"116"}}}`
TransferedEvent	transferringDevice, transferredToDevice `"jsonEvent":{"TransferedEvent":{"primaryOldCall":{"deviceID":"","callID":""},"localConnectionInfo":"","transferringDevice":{"deviceIdentifier":""},"monitorCrossRefID":"","cause":"","transferredToDevice":{"deviceIdentifier":""},"transferredConnections":{"connection`

```
ListItem":[{"newConnection":{"deviceID":"","
callID":""},"oldConnection":{"deviceID":"","
callID":""},"endpoint":{"deviceID":""}},{"ne
wConnection":{"deviceID":"","callID":""},"en
dpoint":{"deviceID":""}},{}]}}}
```


(*) optional

5.2 GetInstantEvents

Description
Fetches all events for the given device object. The function returns events only if the device has an active monitor and events occur. This function returns an answer immediately after send. If there is no event stored for the requested device, an empty answer will be transferred to the sender. The Web Services hold events for 30 seconds. After that the events will be erased. It is like a polling mechanism. It is the job of the application/programmer to take care of sending the right amount of GetInstantEvents commands. It is absolute necessary to use MonitorStart before otherwise the GetInstantEvents function will not receive any events. This function is often used for detecting incoming calls.

Parameter	Type	Description
deviceObject	**deviceID**	The phone where monitoring should start stop.
Filter	**eventtype**	Optional. Get only events matching the filter.

Event types supported by the filter:

- ServiceInitiatedEvent
- OriginatedEvent
- DeliveredEvent
- ConnectionClearedEvent
- EstablishedEvent
- FailedEvent
- TransferedEvent
- HookSwitchEvent

Return Values

negative

Error Code	Response Type
1	deviceHasNoMonitor(no Monitor started)

Return Values Example:

positive


```
{"error":0,"events":[{"jsonEvent":{"ConnectionClearedEvent":{"localConnectionInfo":"nul
l","monitorCrossRefID":"0001","cause":"normalClearing","releasingDevice":{"deviceIde
```

ntifier":"101_103"},"droppedConnection":{"deviceID":"103","callID":"00000028"}}},"receivedAt":1226062777,"calledID":"103","deviceID":"103","callerID":"101","type":"ConnectionClearedEvent","EventForDevice":"103","callID":"00000028"}]}

If there is no event there will be an answer like this, with no event information:

{"error":0,"events":[]}

The result is of the type “JSON string”, containing an array of events or an empty array. If there were no events the array is empty and in that case just poll again. The client application has to evaluate this JSON string.

Example http-request

http://[Web Services IP address]:8801/cgi-bin/gadgetapi?cmd=GetEvents&deviceObject=103

Example http-request Description

Gets the events from PBX for device 103.

Events

The same definition are valid like at chapter 5.1 GetEvents.

5.3 MonitorStart

Description
Starts monitoring for a phone, in order to get information about its events. These events will be collected and can be retrieved with the GetEvents or GetInstantEvents request.
The monitor will be active for until it will be stopped.

Parameter	Type	Description
deviceObject	**deviceID**	The phone which has to be monitored

Return Values

positive

Error Code	Response Type
0	MonitorStartResponse

negative

Error Code	Response Type
1	deviceHasNoMonitor

Example http-request

http://[Web Services IP address]:8801/cgi-bin/gadgetapi?cmd=MonitorStart&deviceObject=103

Example http-request Description

Starts monitoring for device 103.

Example possible, positive JSON response from Web Services

```
{"crossRefID":"0001","error":0,"respType":"MonitorStartResponse","MonitorCount":3}
```

5.4 MonitorStop

Description
Stops monitoring for a phone.

Parameter	Type	Description
deviceObject	**deviceID**	The phone where monitoring should stop.

Return Values

positive

Error Code	Response Type
0	MonitorStopResponse

negative

Error Code	Response Type
1	deviceHasNoMonitor

Example http-request

http://[Web Services IP address]:8801/cgi-bin/gadgetapi?cmd=MonitorStop&deviceObject=103

Example http-request Description

Stops monitoring for device 103.

Example possible, positive JSON response from Web Services

```
{"crossRefID":"0001","error":0,"respType":"MonitorStopResponse","MonitorCount":2}
```

5.5 GetFilter

Format: cmd=GetFilter&user=100

Returns the Filter list associated with the user in a session.
Response:
<Getfilter>
CSTA
EVM
PRESENCE
DIALPARAM
CRE
VMS
</GetFilter>
The list is new line separated. CRE is Call Related Events, not CSTA based, VMS is VoiceMailSystem (OSO).

5.6 SetFilter

Format: cmd=SetFilter&user=100&filter={CSTA,EVM,PRESENCE,DIALPARAM,CRE}

Sets the event fillter for a user within an session. The values passed in 'filter' are comma separated. If none is specified all events are filtered (nothing is sent).
It controls the events for a session regardless which or how many devices are monitored.
CRE is Call Related Events, not based on CSTA
Response:
<SetFilter/>

5.7 Error Codes and Responses

Commands can return either execution specific errors and results (see specific commands) or general ones. General ones can be:

<ERROR>

<REASON></RESON>

</ERROR>

Where RESON can be one of:

- MISSING_PARAM, When at least 1 parameter is missing
- INVALID_PARAM, When the value of a parameter is incorrect
- UNKOWN_CMD, Command not known
- NOT_LGGED_IN, client not logged in
- WOULD_BLOCK, When there is no LAC server address is configured
- NOPERM, No Permission to execute this command for the given user
- NOKEY, No License key
- NOCONN, No connection to LAC

5.8 Event extensions

These are sent though the event channel. Use GetInstantEvents to fetch them. Use SetFilter to narrow the events type to your need.

Dialparam change:

{DialParam:{dialout:”0”,country:”49”,nat:”0”,intl:”00”, area:”2302”,disa:”01234”},type:”DialParam”}

Presence change event:

Presnum is a presence state number

Ptext is optional

{jsonEvent:{PresenceEvent:{deviceID:”102”,newstate:presnum,ptext:”anything”}},
receivedAt:time,EventForDevice:”100”,type:”PresenceEvent”}

Snapshot event:

{Device:{state:”connected/null”,callcnt:callnum,deviceID:”100”},Server:”{LAC:0/1,EVM:0/1},type:”State”}

Server tag shows the connection information. 1 if LAS connected to the component, 0 if not.

5.9 ListSession

After successful login the client may request a session list for the actual user. The list shows where the user is logged from and with which client. The time elapsed since the last request is also sent.

Format: cmd=ListSession&user=100&stype={any,wbmc,widget,OpenStage}

Response:

<SESSION_LIST>

<SESSION>

<ID>session ID</ID>

<TYPE>client type</TYPE> <!—widget,wbmc,OpenStage,none>

<IP> ipaddr </IP>

<IDLE>seconds</IDLE>

</SESSION>

</SESSION_LIST>

5.10 KickSession

Tries to kick one or more sessions. If stype is given all sessions with that type will be kicked(except that one which is calling KickSession). If session IDs are given with comma separated list sessions with the IDs will be removed. The user should be logged in to execute this command. The actual IDs of sessions belonging to a user can be retreived with ListSession.

Format: cmd=KickSession&user=100&stype={any,wbmc,widget,OpenStage}&sessoins=id1,id2,…

Response: <SESSION_KICK/>

6 Phonebooks

There is a general interface to all types of phonebook. The following interfaces exist but not all phonebooks implement everything.

The phonebook can contain several entries for a user. Currently the following entries are supported (a phonebook type can support less):

- Surname
- Firstname
- Phone
- Phone2
- Phone3
- Email
- For requests the 'book' parameter is mandatory. It indicates which phonebook type the access will be made to. See below for details.

6.1 Phonebook search

Reads the phonebook given with the "book" argument for a user. The command works as previously except that it takes an additional optional argument for the interface version. When it's missing or zero the Old V1 interface will be used, when 2 then V2 interfae is used. Here the difference is covered only.

If the "name" parameter is present then the firstname and surname parameters are ignored and the specified string is searched in the lastname and in the firstname as well with a logical OR operator.

Format:

Cmd=PhBookSearch&user=100&book={int|all|edir|pdir|ext}[&surname=Smith&firstname=John&name=Joe&start=1&maxcnt=100&iver={0|1}]

Response:

<PHBOOK cnt="100">

<ITEM>

… Phonebook related tags …

<PRESENCE>

<USER></USER>

<STATE></STATE> <!—presence state

<PTEXT><PTEXT> <!—validTill time except Office where it is the CallMeNumber if set

<PHSTATE>callstate</PHSTATE>

</PRESENCE>

</ITEM>

</PHBOOK>

Phonebook types:

- int: Device list
- edir: Speed dials
- pdir: Personal directory
- all: All phonebooks

The Presence part is identical to the response of PresenceGetXML.

6.2 Phonebook detail

Cmd=PhBookDetail&book=type&user=x&id=x

This command gives the details about a phonebook entry and returns the following on success:

<ITEM>

<ID>%d</ID> ID for the entry (for further operations)

<USER> user </USER>

<SURNAME> surname </SURNAME>

<FIRSTNAME> firstname </FIRSTNAME>

<PHONE> 01234567 </PHONE>

<PHONE2> 01234567 </PHONE2>

<PHONE3> 01234567 </PHONE3>

<EMAIL> email </EMAIL>

</ITEM>

6.3 Phonebook delete

Cmd=PhBookDelete&book=type&user=x&id=x

The phonebook entry in user's favourites identified by id 'x' will be removed.

6.4 Phonebook add

Cmd=PhBookAdd&book=type&user=x{&firstname=y}{&surname=z}{&email=v}{&phone=a}{&phone2=b}{&phone3=c}

The command will add the entry with the parameters supplied to the user's favourites. If a parameter is not supplied it will be empty string.

6.5 Phonebook update

Cmd=PhBookUpdate&book=type&user=x&id=x{&firstname=y}{&surname=z}{&email=v}{&phone=a}{&phone2=b}{&phone3=c}

The command will add the entry with ID 'x' in the user's phonebook. Parameters supplied will be updated. Those parameters that do not appear will be untouched.

7 User Presence

7.1 SetPresence

Format: http://ipaddr/cgi-bin/gadgetapi?cmd=SetPresence&user=100&presence=state_id&ptext=time

The state_id (presence state) can be:

Numerical STATE	Meaning
1	Office
2	Meeting
3	Sick
4	Break
5	GoneOut
6	Holiday
7	Lunch
8	Home
9	DND (*Do Not Disturb)

The time format is: yyyy-mm-dd hh:mmZ, e.g. 2011-10-15 08:30Z.

7.2 SetCallMe

Format: http://ipaddr/cgi-bin/gadgetapi?cmd=SetCallMe&user=100[&callme=phonenumber]

Sets the CallMe state for the user if the CallMe number is specified. Otherwise it will deactivate the CallMe state (presence changes to Office).

7.3 GetPresence

Format: http://ipaddr/cgi-bin/gadgetapi?cmd=GetPresence&user=100

The server returns the presence state in the same format listed above except that the 'text' field indicates the time till the state is valid (except Office where it indicates the CallMeNumber if set):

{"error":0, "status":presenceid, "text":"time",phstate:"callstate"}

Callstate can be one of:

Ringing

Hold

Established

Idle

For the "presenceid" see the GetPresenceXML command.

7.4 GetPresenceXML

Format: http://ipaddr/cgi-bin/gadgetapi?cmd=GetPresenceXML&user=100,101,...

The server returns the presence state for the listed devices(users) in the format:

<PRESENCE_LIST>

<PRESENCE>

<USER></USER>

<STATE></STATE> <!—presence state

<PTEXT><PTEXT> <!—validTill time except Office where it is the CallMeNumber if set

<PHSTATE>callstate</PHSTATE>

</PRESENCE>

</PRESENCE_LIST>

See callstate in GetPresence

7.5 PresenceDBGet

This method gets the forwarding target for a given presence state.

Cmd=PresenceDBGet&user=100&state=presencestate

Returns:
<PRESENCE_DB>
<PRESENCE>
<FORWARD></FORWARD>
<STATE>number</STATE>
</PRESENCE>
</PRESENCE_DB>

7.6 PresenceDBSet

This method sets the forwarding target for a given presence state. When the presence state is set, the server will automatically set the forwarding.

Cmd=PresenceDBSet&user=100&state=presencenum&forwarding=number
Returns:
<PRESENCEDBSET/>

8 User Journal

8.1 JournalRead

Format:
cmd=JournalRead&user=100&[maxcnt=v|start=u||dir={in|out}|est={yes|no}][&startdtime=YYYY-MM-DD HH:MM:SS][&enddtime=YYYY-MM-DD]

start and maxcnt will work the same way as in the PhBookSearch command.

Response, list:
<JOURNAL cnt="x">
<ITEM>

<JID></JID> <!-- Journal ID>
<DIR>in|out</DIR>
<EST>yes|no</EST>
<CALLER>1234567</CALLER>
<CALLED>12</CALLED>
<BEGIN>YYYY-MM-DD HH:MM:SS</BEGIN>
<END>YYYY-MM-DD HH:MM:SS</END>
<DURATION>seconds</DURATION>
<PHENTRY>
<ID>1</ID>
<SURNAME>Name1</SURNAME>
<FIRSTNAME>Name2</FIRSTNAME>
<TYPE>pdir</TYPE>
</PHENTRY>
</ITEM>
</JOURNAL>

JournalRead will try to resolve the called or caller number based on the direction using the Device list, Speed Dials, and the user's Personal Directory. If there is a matching entry – **and only if there is one!** – the <PHENTRY> tag will be included containing the name, the Phonebook ID and the phonebook type where it was found.

The algorithm tries to find the best matching number. It is possible to specify start and/or end date –time. In this case the list will contain only calls that falls into the specified date range.
List is returned in ascending order by date.

8.2 JournalGroupByDate

Format:
cmd=JournalGroupByDate&user=100&[maxcnt=v|start=u||dir={in|out}|est={yes|no}][&startdtime=YYYY-MM-DD HH:MM:SS&enddtime=YYYY-MM-DD]

start and maxcnt will work the same way as in the PhBookSearch command.

Response, list:
<JOURNALGROUP cnt="x">
<ITEM>
<CNT></CNT> <!—how many calls received on the given day
<BEGIN>YYYY-MM-DD 00:00:00</BEGIN>
</ITEM>
</JOURNALGROUP>

It will return a list off dates with the number of calls on that days matching the given constraints.
The interface works similar as the phonebook, fragmentation is possible.
Specifying the start and end date is also possible but here both must be specified. Works as in the JournalRead case.
List is returned in ascending order by date.

9 Appendix

9.1 Abbreviations

AC	Application Cookbook
API	Application Programming Interface
CGI	Common Gateway Interface
CSTA	Computer Supported Telecommunications Application
CSP	CSTA Service Provider
DB	Database
EVM	Entry Voice Mail
WS	Web Services
WSI	Web Services Interface
WSI Reference	Web Services Interface Reference
HFA	HiPath Feature Access
HOOME	HiPath OpenOffice ME
Html	Hypertext Markup Language http Hypertext Transfer Protocol
https	Hypertext Transfer Protocol Secure
IP	Internet Protocol
IP-address	Internet Protocol address
JRE	Java™ Runtime Environment
LAS	Lightweight Application Server
MB	MegaByte
Mbit/s	Megabit pro Sekunde
ME	Medium Enterprise
PBX	Private Branch Exchange
PWD	password
RAM	Random Access Memory
SDK	Software Development Kit
SIP	Session Initiation Protocol
SP2	Service Pack 2
v	version
WBM	Web Based Management

About Unify

Unify is one of the world's leading communications software and services firms, providing integrated communications solutions for approximately 75 percent of the Fortune Global 500. Our solutions unify multiple networks, devices and applications into one easy-to-use platform that allows teams to engage in rich and meaningful conversations. The result is a transformation of how the enterprise communicates and collaborates that amplifies collective effort, energizes the business, and enhances business performance. Unify has a strong heritage of product reliability, innovation, open standards and security.

Unify.com

UNIFY Harmonize your enterprise