Cisco TelePresence TelePresence Server API 2.2

Product Programming Reference Guide

D14867

August 2011

Contents

Introduction

This document accompanies the latest version of the remote management API for the Cisco TelePresence Server software (respectively referred to as API and TelePresence Server in this document). The following Cisco TelePresence products support this API when they are running TelePresence Server version 2.2 and later:

- Cisco TelePresence Server MSE 8710
- Cisco TelePresence Server 7010

XML-RPC implementation

API calls and responses are implemented using the XML-RPC protocol. This simple protocol does remote procedure calling using HTTP (or HTTPS) as the transport and XML as the encoding. It is extremely simple although it does still allow for complex data structures. XML-RPC is stateless and is not platform-dependent; it was chosen in favor of SOAP (Simple Object Access Protocol) because of its simplicity.

Your application must either regularly poll the device or continually listen to the device - if it is configured to publish feedback events - if you want it to monitor the device's activity.

The API implements all parameters and returned data as **<struct>** elements, each of which is explicitly named. For example, **device.query** returns (amongst other data) the current time as:

```
<member>
  <name>currentTime</name>
  <value><dateTime.iso8601>20110121T13:31:26<dateTime.iso8601></value>
</member>
```

rather than simply

```
<dateTime.iso8601>20110121T13:31:26<dateTime.iso8601>
```

Note: Unless otherwise stated, assume strings have a maximum length of 31 characters.

Refer to the XML-RPC specification[1] for more information.

Transport protocol

The device implements HTTP/1.1 as defined by RFC 2616[2]. It expects to receive communications over TCP/IP connections to port 80 (default HTTP port) or port 443 (default HTTPS port).

Your application should send HTTP POST messages to the URL defined by path **/RPC2** on the device's IP address, for example **https://10.0.0.53/RPC2**.

You can configure the device to receive HTTP and HTTPS on non-standard TCP port numbers if necessary, in which case append the non-standard port number to the IP address.

Considering API overhead when writing applications

Every API command that your application sends incurs a processing overhead within the device's own application. The exact amount of overhead varies widely with the command type and the parameters sent. It is important to bear this in mind when designing your application's architecture and software. If the device receives a high number of API commands every second, its overall performance could be seriously impaired – in the same way that it would be if several users accessed it simultaneously via the web interface.

There is a limit on the number of simultaneous requests that the TelePresence Server can process. If your client application receives a related HTTP error such as **`503 Service Unavailable`** then it should retry the request.

For this reason, the best architecture is a single server running the API application and sending commands to the device. If multiple users need to use the application simultaneously, provide a web interface on that server or write a client that communicates with the server. The server would then manage the clients' requests and send API commands directly to the device. Implement some form of control in the API application on your server to prevent the device being overloaded with API commands. This provides much more control than having the clients send API commands directly and will prevent the device's performance being impaired by unmanageable numbers of API requests.

Furthermore, the API is designed to have as little impact as possible on the network when responding to requests. The device's responses do not routinely include data that is not relevant, or empty data structures where the data is not available. Your application should take responsibility for checking whether the response includes what you expected, and you should design it to gracefully handle any situations where the device does not respond with the expected data.

API overview

Encoding

Your application can encode messages as ASCII text or as UTF-8 Unicode. If you do not specify the encoding, the API assumes ASCII encoding. You can specify the encoding in a number of ways:

Specify encoding with HTTP headers

There are two ways of specifying UTF-8 in the HTTP headers:

- Use the **`Accept-Charset: utf-8`** header
- Modify the **`Content-Type`** header to read **`Content-Type: text/xml; charset=utf-8`**

Specify encoding with XML header

The **`<?xml>`** tag is required at the top of each XML file. The API will accept an encoding attribute for this tag; that is, **`<?xml version="1.0" encoding="UTF-8"?>`**.

Authentication

Note: Authentication information is sent using plain text and should only be sent over a trusted network.

The controlling application must authenticate itself on the device as a user with administrative privileges. Also, because the interface is stateless, every call must contain authentication parameters:

`authenticationUser`

Type: **string**

Name of a user with sufficient privilege for the operation being performed. The name is case sensitive.

`authenticationPassword`

Type: **string**

The password that corresponds with the given **`authenticationUser`**. The API ignores this parameter if the user has no password. This behavior differs from the web interface, where a blank password must be blank.

Message flow

The application initiates the communication and sends a correctly formatted XML-RPC command to the device.

Example command

```
<?xml version='1.0' encoding='UTF-8'?>
```

```
<methodCall>
  <methodName>recording.delete</methodName>
    <params>
      <param>
        <value>
          <struct>
            <member>
              <name>authenticationPassword</name>
              <value><string></string></value>
            </member>
            <member>
              <name>recordingId</name>
              <value><int>101</int></value>
            </member>
            <member>
              <name>authenticationUser</name>
              <value><string>admin</string></value>
            </member>
          </struct>
        </value>
      </param>
    </params>
</methodCall>
```

Assuming the command was well formed, and that the device is responsive, the device will respond in one of these ways:

- With an XML **`methodResponse`** message that may or may not contain data, depending on the command.
- With an XML **`methodResponse`** that includes only a fault code message.

Example success

```
<?xml version="1.0"?>
  <methodResponse>
    <params>
      <param>
        <value>
          <struct>
            <member>
              <name>status</name>
              <value>
                <string>operation successful</string>
              </value>
            </member>
          </struct>
        </value>
      </param>
    </params>
  </methodResponse>
```

Example fault code

```
<?xml version="1.0"?>
 <methodResponse>
  <fault>
    <value>
      <struct>
        <member>
          <name>faultCode</name>
            <value>
              <int>22</int>
            </value>
        </member>
        <member>
          <name>faultString</name>
            <value>
              <string>no such recording</string>
            </value>
        </member>
      </struct>
    </value>
  </fault>
 </methodResponse>
```

API reference

This is a list of the API calls supported by the TelePresence Server. For each API call, the following information is provided where applicable:

- Description of the call's function and status
- Accepted parameters
- Returned parameters, structure formats and data types
- Deprecated parameters

Click the call name to read a detailed description of the call.

Deprecations

Deprecated parameters

The following parameters are deprecated in this version of the API. Update your applications to use the replacement parameters instead; these parameters may not be supported in future releases.

In calls that can still accept the deprecated parameters, take care to send only the deprecated parameter or the replacement parameter; not both.

Deprecated parameter	Replaced by	The affected calls
permanent	persistent	conference.create
conferenceID	conferenceGUID	conference.invite conference.create conference.delete conference.enumerate conference.senddtmf conference.sendmessage conference.sendwarning conference.set conference.status conference.uninvite participant.enumerate
participantList (string)	**participants** (array)	conference.invite
participantID	participantGUID	conference.invite conference.senddtmf conference.sendmessage conference.status participant.enumerate participant.set participant.tidylayout
omitID	omitGUID	conference.senddtmf
endpointType	endpointCategory	conference.status
participantListID	participantListGUID	conference.uninvite
roundTableEnable	oneTableMode	conference.set conference.status

cdrlog.enumerate

Status: **active**

This call allows the calling application to download CDR log data without having to return the entire CDR log. The call returns a subset of the CDR log based on the optional **filter**, **index** and **numEvents** parameters.

The TelePresence Server holds up to 2000 records in memory. It does not permanently retain these, so we recommend that your application either makes regular enumerate calls or triggers enumerate calls upon receiving the **cdrAdded** feedback event.

Accepts:

Optional:

filter

Type: **array**

An array of strings, each of which is the name of an event type by which to filter the response. If the array is omitted, all event types are returned.

For the TelePresence Server, the call can request any / all of the following event types:

- conferenceStarted
- conferenceActive
- participantConnected
- participantJoined
- participantMediaSummary
- participantLeft
- participantDisconnected
- conferenceInactive
- conferenceFinished

index

Type: **integer**

Index from which to get events. The device returns the nextIndex so the application can use it to retrieve the next enumeration of CDR data.

If index is omitted, negative, or greater (by 2 or more) than the highest index, then the device will enumerate events from the beginning of the CDR log.

numEvents

Type: **integer**

Specifies maximum number of events to be returned per enumeration. If omitted (or not between 1 - 20 inclusive), a maximum of 20 events will be returned per enumeration.

Returns:

The response provides reference information such as time and log position, and an array of events that meet the parameters provided in the call.

startIndex

Type: **integer**

Either the index provided, or if that is lower than the index of the first record the device has, it will be the first record it does know about. In this case, comparing the **startIndex** with the index provided gives the number of dropped records.

nextIndex

Type: **integer**

Revision number of the data being provided, reusable in a subsequent call to the API.

eventsRemaining

Type: **boolean**

Whether there is data remaining after this. Provided to avoid putting all data in a single call.

`currentTime`

Type: **dateTime.iso8601**

The system's current time (UTC).

`events`

Type: **array**

List of the new events; these are structures with some common fields (time, type, index) and other fields specific to the event type.

`events` array

The following parameters are common to all CDR log events. The array also contains infomation members specific to each event. The CDR log reference guide[3] contains details of the TelePresence Server event types.

Note: The CDR log reference guide describes the CDR log in its XML form, as downloaded in **cdr_log.xml** via the web interface. When the same events are enumerated with this call, the event type names use camelCase for multiple words rather than using underscores. For example, **`conference_started`** in **cdr_log.xml** is the same event type as **`conferenceStarted`** in this array.

If there are no events to enumerate, the **`events`** array is returned empty.

`time`

Type: **dateTime.iso8601**

The date and time when the event was logged.

Value	Description
20110119T13:52:42	yyyymmddThh:mm:ss

`type`

Type: **string**

The name of the event type.

`index`

Type: **integer**

A number that identifies the position of the item in context with similar items.

cdrlog.query

Status: **active**

This call queries for statistics about the CDR log.

This call takes no parameters.

Returns:

`firstIndex`

Type: **integer**

The index of the oldest stored event.

`numEvents`

Type: **integer**

The total number of events stored.

conference.create

Status: **active**

Creates a conference with the specified name and other supplied parameters, and returns the unique identifier of the new conference.

Accepts:

Required:

`conferenceName`

Type: **string**

The name that refers to the conference that is the subject of your call or the response from the TelePresence Server.

Optional:

Use `persistent` instead of `permanent` to define conference persistence, even though the TelePresence Server will accept either.

`persistent`

Type: **boolean**

Defines whether the conference persists after all participants leave. Persistent conferences are stored in the configuration file and thus will persist through a device restart.

Value	Description
true	The conference persists, irrespective of participants leaving, until it is explicitly deleted.
false	The conference is deleted 30 seconds after all participants have left, or when `duration` expires (if it is set).

`permanent`

Type: **boolean**

Deprecated. Use **`persistent`** instead.

Defines whether the conference persists after all participants leave. Without this option any conferences will be automatically deleted after 30 seconds, or when **`duration`** expires (if it is set).

`locked`

Type: **boolean**

Defines whether the conference is locked.

Endpoints can not join a locked conference but the conference can invite them in.

Value	Description
true	The conference is locked.
false	The conference is not locked.

`lockDuration`

Type: **integer**

The period of time (in seconds) from now until the conference lock expires. Requires that **`locked`** is **`true`** and ignored otherwise.

`numericId`

Type: **string**(< 32 characters)

Used for registration with H.323 gatekeeper / SIP registrar, and to dial in to the conference.

`registerWithGatekeeper`

Type: **boolean**

Defines whether or not this item registers its **`numericId`** with the H.323 gatekeeper.

`registerWithSIPRegistrar`

Type: **boolean**

Defines whether or not this item registers its **`numericId`** with the SIP registrar.

`tsURI`

Type: **string**

The address that Cisco TelePresence System T3 systems use to make API calls to the TelePresence Server.

This string must take the form **`[<protocol>://]<address>[:<port>]`**, for example, **`http://mytps:80`**. If supplied, this URI will be passed to all T3 systems in the conference via TString. If not explicitly supplied, the TelePresence Server will create a **`tsURI`** based on its IP address.

http and **https** protocols are supported. The TelePresence Server does not assume protocol or port information if the application does not supply them in this string.

h239ContributionEnabled

Type: **boolean**

Defines whether the conference allows content contribution. This parameter controls whether content may be contributed via any of the supported content protocols; it is not limited to H.239.

useLobbyScreen

Type: **boolean**

Defines whether the conference shows the lobby screen.

lobbyMessage

Type: **string**

The lobby screen message.

useWarning

Type: **boolean**

Defines whether the conference sends 'This conference is about to end' warning.

audioPortLimit

Type: **integer**

The limit on the number of audio ports this conference may allow.

videoPortLimit

Type: **integer**

The limit on the number of video ports this conference may allow.

duration

Type: **integer**

Period of time (in seconds) until the conference ends and is deleted.

This parameter is not allowed if **persistent** is **true**.

Returns:

conferenceGUID

Type: **string**

Globally unique identifier of the conference.

conferenceID

Type: **integer**

Deprecated. Use **conferenceGUID** instead.

Unique conference identifier.

conference.delete

Status: **active**

Deletes the specified conference.

Accepts:

Required:

To identify the conference, use **conferenceGUID** instead of **conferenceID**, not both.

conferenceGUID

Type: **string**

Globally unique identifier of the conference.

conferenceID

Type: **integer**

Deprecated. Use **conferenceGUID** instead.

Unique conference identifier.

conference.enumerate

Status: **active**

Requests information about all the conferences on the TelePresence Server. The full enumeration response may require multiple calls.

Accepts:

Optional:

enumerateID

Type: **integer**

Enumerate calls may return many results so all of them will accept this parameter and may include this parameter in the response.

If the response includes an enumerateID, the application should pass the ID to the subsequent enumerate call to retrieve the next set of results. If the response does not include an enumerateID, there are no more results in the enumeration.

If the application omits the enumerateID, the target device will start a new enumeration and return the first set of results.

activeFilter

Type: **boolean**

Value	Description
true	Request only active conferences
false	Request all conferences. This is the default value; it is assumed if you omit the parameter.

Returns:

If there are no conferences to enumerate, then the **conference.enumerate** call does not return the **conferences** array.

Conditional:

enumerateID

Type: **string**

Enumerate calls may return many results so all of them will accept this parameter and may include this parameter in the response.

If the response includes an enumerateID, the application should pass the ID to the subsequent enumerate call to retrieve the next set of results. If the response does not include an enumerateID, there are no more results in the enumeration.

If the application omits the enumerateID, the target device will start a new enumeration and return the first set of results.

conferences

Type: **array**

An array of structs, each of which contains all the returned information about a single conference.

conferences array members

The following information is returned about the enumerated conferences:

conferenceName

Type: **string**

The name that refers to the conference that is the subject of your call or the response from the TelePresence Server.

conferenceGUID

Type: **string**

Globally unique identifier of the conference.

conferenceID

Type: **integer**

Deprecated. Use **conferenceGUID** instead.

Unique conference identifier.

The TelePresence Server returns this parameter, even though it is deprecated, to ensure application compatibility in the short term. We recommend that you use the replacement parameter instead.

active

Type: **boolean**

Value	Description
true	The conference is currently active
false	The conference is currently inactive (e.g. a persistent conference without any active participants, or a non-persistent conference that has not yet started or is empty but not yet deleted)

persistent

Type: **boolean**

Defines whether the conference persists after all participants leave. Persistent conferences are stored in the configuration file and thus will persist through a device restart.

Value	Description
true	The conference persists, irrespective of participants leaving, until it is explicitly deleted.
false	The conference is deleted 30 seconds after all participants have left, or when **duration** expires (if it is set).

locked

Type: **boolean**

Defines whether the conference is locked.

Endpoints can not join a locked conference but the conference can invite them in.

Value	Description
true	The conference is locked.
false	The conference is not locked.

numericId

Type: **string**(< 32 characters)

Used for registration with H.323 gatekeeper / SIP registrar, and to dial in to the conference.

This is an empty string if the parameter is not set.

registerWithGatekeeper

Type: **boolean**

Defines whether or not this item registers its **numericId** with the H.323 gatekeeper.

registerWithSIPRegistrar

Type: **boolean**

Defines whether or not this item registers its **numericId** with the SIP registrar.

h239ContributionEnabled

Type: **boolean**

Defines whether the conference allows content contribution. This parameter controls whether content may be contributed via any of the supported content protocols; it is not limited to H.239.

conference.invite

Status: **active**

Invites the specified participants to the specified conference.

Avoid using the **conferenceID** and **participantList** parameters and use the replacement **conferenceGUID** and **participants** parameters instead.

Accepts:

Required:

To identify the conference, use **conferenceGUID** instead of **conferenceID**, not both.

To identify the participants, use the **participants** array instead of **participantList**, not both.

conferenceGUID

Type: **string**

Globally unique identifier of the conference.

conferenceID

Type: **integer**

Deprecated. Use **conferenceGUID** instead.

Unique conference identifier.

participantList

Type: **string**

Deprecated. Use **participants** array instead.

A comma separated list of participant addresses, with optional extra information.

Value	Description
Example	**10.2.171.232, 10.47.2.246, h323:numericID@domain.com**
Example with type	**10.2.171.232, t3:h323:numericID@domain.com** (specify the endpoint type,followed by a colon, before the protocol)
Example with master	**10.2.171.232, t3:master:h323:numericID@domain.com** (specify **master:** in the prefix; immediately after the endpoint type, if present, and before the protocol)

participants

Type: **array**

An array of structures that represent participants.

participants array

You must include an array of participants in your **conference.invite** call. Each participant must have an **address** parameter. All participant parameters except **address** are optional and the TelePresence Server will use the default value if your call omits them.

address

Type: **string**

The address of the item, e.g. endpoint or gateway; may be hostname, IP address or E.164 number.

You must prefix the address with either **h323:** or **sip:**. If you don't provide a prefix, the TelePresence Server attempts to call the address directly, using H.323 (not via the gatekeeper).

You may provide a comma separated list of up to four addresses if you are inviting a grouped endpoint (requires a third-party interop feature key installed on the TelePresence Server). In this case you should provide a protocol prefix for each address, for example **h323:leftmost_endpoint@domain.com,h323:rightmost_endpoint@domain.com**, but must not supply a type prefix.

type

Type: **string**

Specifies the type of endpoint.

Value	Description
t3	Cisco TelePresence System T3
exp	TANDBERG Experia
cts	Any Cisco TelePresence System 'telepresence' endpoint (1 or 3 screen, e.g. 500, 1300, 3000)

Value	Description
cts1	Cisco TelePresence System single screen 'telepresence' endpoints (e.g. 500 and 1300 series)
cts3	Cisco TelePresence System three screen 'telepresence' endpoints (e.g. 3000 series)

master

Type: **boolean**

Value	Description
true	This endpoint is conference master
false	(default if omitted) This endpoint is not the conference master

oneTableIndex

Type: **integer**

The endpoint's position if it is in a OneTable conference. Applies only if **type** is **t3** or **exp**.

Value	Description
1, 2, 3, or 4	Position index of the endpoint when it is in OneTable mode. The positions increment around the one virtual table in a clockwise manner, when the table is viewed from above. For example, the participant whose index is 2 will appear to be sitting to the left of the participant whose index is 1.

maxBitRate

Type: **integer**

The maximum bitrate, in kbps, in both directions between the TelePresence Server and this participant. The TelePresence Server uses its default setting if your call omits this parameter.

recordingDevice

Type: **boolean**

Value	Description
true	The endpoint is treated as a recording device; it does not feature in the layout and other participants are made aware of its presence by a red dot as appropriate.
false	(default if omitted) The endpoint is a normal endpoint.

dtmf

Type: **string**

DTMF character string to send to this endpoint after connection.

audioContentIndex

Type: **integer**

Defines which endpoint in a group should receive the content and audio. This is a zero-based index that corresponds to the entries provided in the comma separated list of endpoint addresses in **address**.

Value	Description
0	(default) The first address in the address string
n-1	(maximum) The last address in a comma separated string of n addresses

camerasCrossed

Type: **boolean**

Value	Description
true	The cameras of a grouped endpoint are crossed; this is ignored unless this participant is a grouped endpoint, i.e. has multiple **address** parameters
false	(default if omitted) The cameras are not crossed

txAspectRatio

Type: **string**

Overrides the aspect ratio of the layout transmitted to this participant.

Value	Description
only16to9	Force the TelePresence Server to send a widescreen layout (16:9) to the endpoint, overriding any box-wide or per-endpoint settings
only4to3	Force the TelePresence Server to send a 4:3 layout to the endpoint, overriding any box-wide or per-endpoint settings

Returns:

participantList

Type: **array**

Array of participants. Each member of the array is a struct that represents a participant on the TelePresence Server.

participantList array members

The returned **participantList** is an array of successfully invited participants. Note that the member structs of this array are different to those returned in **participantList** by the **conference.status**

call.

Each struct contains the following parameters:

participantGUID

Type: **string**

The GUID of this participant, assigned by the TelePresence Server.

participantID

Type: **integer**

Deprecated. Use `participantGUID` instead.

The unique ID of this participant, assigned by the TelePresence Server.

The TelePresence Server returns this parameter, even though it is deprecated, to ensure application compatibility in the short term. We recommend that you use the replacement parameter instead.

address

Type: **string**

The address of the item, e.g. endpoint or gateway; may be hostname, IP address or E.164 number.

These addresses are as you supplied them in the participants array, to make them easier to compare.

conference.senddtmf

Status: **active**

Sends a DTMF string to some or all participants in the specified conference. You must specify the conference and the DTMF string.

If you don't specify a participant, the string goes to all participants; otherwise, you may specify either a participant who will receive the string or one who will not receive the string.

Accepts:

Required:

To identify the conference, use `conferenceGUID` instead of `conferenceID`, not both.

conferenceGUID

Type: **string**

Globally unique identifier of the conference.

conferenceID

Type: **integer**

Deprecated. Use `conferenceGUID` instead.

Unique conference identifier.

dtmf

Type: **string**

DTMF character string to send to this endpoint after connection.

Optional:

To identify the participant to receive DTMF, use **participantGUID** instead of **participantID**, not both. Alternatively, to identify the participant who won't receive DTMF, use **omitGUID** instead of **omitID**, not both.

participantGUID

Type: **string**

The GUID of this participant, assigned by the TelePresence Server.

If you supply this parameter, the DTMF string will be sent to this participant only.

participantID

Type: **integer**

Deprecated. Use **participantGUID** instead.

The unique ID of this participant, assigned by the TelePresence Server.

omitGUID

Type: **string**

A **participantGUID**. Prevents this participant from receiving the DTMF string specified in **dtmf**.

If you supply this parameter, the DTMF string will be sent to all participants except this one. If **participantGUID** is present, **omitGUID** is ignored.

omitID

Type: **integer**

Deprecated. Use **omitGUID** instead.

A **participantID**. Prevents this participant from receiving the DTMF string specified in **dtmf**.

conference.sendmessage

Status: **active**

Sends a message to all participants in the specified conference. You must specify the conference and the message.

If you choose to specify a participant, the message will only go to that participant.

Accepts:

Required:

To identify the conference, use **conferenceGUID** instead of **conferenceID**, not both.

conferenceGUID

Type: **string**

Globally unique identifier of the conference.

conferenceID

Type: **integer**

Deprecated. Use **conferenceGUID** instead.

Unique conference identifier.

message

Type: **string**

Message to send to conference.

Optional:

To identify a participant, use the **participantGUID** instead of **participantID**, not both.

participantGUID

Type: **string**

The GUID of this participant, assigned by the TelePresence Server.

participantID

Type: **integer**

Deprecated. Use **participantGUID** instead.

The unique ID of this participant, assigned by the TelePresence Server.

position

Type: **integer**

Defines where the message displays on the layout.

Value	Description
1,2, or 3	The message displays near the top of the layout; aligned to the left, center, or right respectively.
4, 5 (default), or 6	The message displays in the middle of the layout; aligned to the left, center, or right respectively.

Value	Description
`7`, `8`, or `9`	The message displays near the bottom of the layout; aligned to the left, center, or right respectively.

`duration`

Type: **integer**

Period of time (in seconds) for which the message is displayed to participants. Default is `30`.

conference.sendwarning

Status: **active**

Sends the 'conference is about to end' warning to all the participants in the specified conference.

Accepts:

Required:

To identify the conference, use `conferenceGUID` instead of `conferenceID`, not both.

`conferenceGUID`

Type: **string**

Globally unique identifier of the conference.

`conferenceID`

Type: **integer**

Deprecated. Use `conferenceGUID` instead.

Unique conference identifier.

conference.set

Status: **active**

Edit the configuration of the specified conference.

Accepts:

Required:

To identify the conference, use `conferenceGUID` instead of `conferenceID`, not both.

`conferenceGUID`

Type: **string**

Globally unique identifier of the conference.

conferenceID

Type: **integer**

Deprecated. Use **conferenceGUID** instead.

Unique conference identifier.

Optional:

To set up one table mode, use **oneTableMode** instead of **roundTableEnable**, not both.

numericId

Type: **string**(< 32 characters)

Used for registration with H.323 gatekeeper / SIP registrar, and to dial in to the conference.

registerWithGatekeeper

Type: **boolean**

Defines whether or not this item registers its **numericId** with the H.323 gatekeeper.

registerWithSIPRegistrar

Type: **boolean**

Defines whether or not this item registers its **numericId** with the SIP registrar.

roundTableEnable

Type: **boolean**

Deprecated. Use **oneTableMode** instead.

Defines whether the conference is in round table mode.

If you supply both **roundTableEnable** and **oneTableMode**, then the TelePresence Server will use **oneTableMode** without returning an error.

oneTableMode

Type: **integer**

Value	Description
0	oneTableMode off
1	4 person oneTableMode
2	2 person oneTableMode

h239ContributionEnabled

Type: **boolean**

Defines whether the conference allows content contribution. This parameter controls whether content may be contributed via any of the supported content protocols; it is not limited to H.239.

locked

Type: **boolean**

Defines whether the conference is locked.

Endpoints can not join a locked conference but the conference can invite them in.

Value	Description
true	The conference is locked.
false	The conference is not locked.

lockDuration

Type: **integer**

The period of time (in seconds) from now until the conference lock expires. Requires that **locked** is **true** and ignored otherwise.

duration

Type: **integer**

Period of time (in seconds) until the conference ends and is deleted.

This parameter is not allowed if **persistent** is **true**.

You can pass a negative value to clear a previously set **duration**.

audioPortLimitSet

Type: **boolean**

Defines whether the **audioPortLimit** is applied.

Value	Description
true	Limits the number of audio ports to the value in **audioPortLimit**
false	**audioPortLimit** is ignored if it is present

You **must** provide an **audioPortLimit** if you set **audioPortLimitSet** to **true**. If you set it **false**, the call clears the existing **audioPortLimit**.

audioPortLimit

Type: **integer**

The limit on the number of audio ports this conference may allow.

videoPortLimitSet

Type: **boolean**

Defines whether the **videoPortLimit** is applied.

Value	Description
true	Limits the number of video ports to the value in **videoPortLimit**
false	**videoPortLimit** is ignored if it is present

You **must** provide a **videoPortLimit** if you set **videoPortLimitSet** to **true**. If you set it **false**, the call clears the existing **videoPortLimit**.

videoPortLimit

Type: **integer**

The limit on the number of video ports this conference may allow.

useLobbyScreen

Type: **boolean**

Defines whether the conference shows the lobby screen.

lobbyMessage

Type: **string**

The lobby screen message.

useWarning

Type: **boolean**

Defines whether the conference sends 'This conference is about to end' warning.

conference.status

Status: **active**

Reports the current status of the specified conference and its participants.

Accepts:

Required:

To identify the conference, use **conferenceGUID** instead of **conferenceID**, not both.

conferenceGUID

Type: **string**

Globally unique identifier of the conference.

conferenceID

Type: **integer**

Deprecated. Use **conferenceGUID** instead.

Unique conference identifier.

Optional:

enumerateID

Type: **string**

Enumerate calls may return many results so all of them will accept this parameter and may include this parameter in the response.

If the response includes an enumerateID, the application should pass the ID to the subsequent enumerate call to retrieve the next set of results. If the response does not include an enumerateID, there are no more results in the enumeration.

If the application omits the enumerateID, the target device will start a new enumeration and return the first set of results.

Returns:

conferenceGUID

Type: **string**

Globally unique identifier of the conference.

conferenceID

Type: **integer**

Deprecated. Use **conferenceGUID** instead.

Unique conference identifier.

The TelePresence Server returns this parameter, even though it is deprecated, to ensure application compatibility in the short term. We recommend that you use the replacement parameter instead.

active

Type: **boolean**

Value	Description
true	The conference is currently active
false	The conference is currently inactive (e.g. a persistent conference without any active participants, or a non-persistent conference that has not yet started or is empty but not yet deleted)

persistent

Type: **boolean**

Defines whether the conference persists after all participants leave. Persistent conferences are stored in the configuration file and thus will persist through a device restart.

Value	Description
true	The conference persists, irrespective of participants leaving, until it is explicitly deleted.
false	The conference is deleted 30 seconds after all participants have left, or when **`duration`** expires (if it is set).

`duration`

Type: **integer**

Period of time (in seconds) until the conference ends and is deleted.

This parameter is not allowed if **`persistent`** is **`true`**.

`locked`

Type: **boolean**

Defines whether the conference is locked.

Endpoints can not join a locked conference but the conference can invite them in.

Value	Description
true	The conference is locked.
false	The conference is not locked.

`lockDuration`

Type: **integer**

The period of time (in seconds) from now until the conference lock expires. Requires that **`locked`** is **`true`** and ignored otherwise.

`roundTableEnable`

Type: **boolean**

Deprecated. Use **`oneTableMode`** instead.

Defines whether the conference is in round table mode.

The TelePresence Server returns this parameter, even though it is deprecated, to ensure application compatibility in the short term. We recommend that you use the replacement parameter instead.

`oneTableMode`

Type: **integer**

Value	Description
0	oneTableMode off
1	4 person oneTableMode
2	2 person oneTableMode

h239ContributionID

Type: **integer**

The **participantID** of the endpoint that is contributing H.239 content. Zero if there is no H.239 contribution.

portsVideoFree

Type: **integer**

Count of the currently unused video ports.

Zero if the conference is inactive.

portsAudioFree

Type: **integer**

Count of the currently unused audio ports.

Zero if the conference is inactive.

portsContentFree

Type: **integer**

Count of the currently unused content ports.

Zero if the conference is inactive.

numericId

Type: **string**(< 32 characters)

Used for registration with H.323 gatekeeper / SIP registrar, and to dial in to the conference.

This is an empty string if the parameter is not set.

registerWithGatekeeper

Type: **boolean**

Defines whether or not this item registers its **numericId** with the H.323 gatekeeper.

registerWithSIPRegistrar

Type: **boolean**

Defines whether or not this item registers its **numericId** with the SIP registrar.

recording

Type: **boolean**

True if this conference is being recorded by a recording device specified in **conference.invite**.

audioPortLimitSet

Type: **boolean**

Defines whether the **audioPortLimit** is applied.

Value	Description
true	Limits the number of audio ports to the value in **audioPortLimit**
false	**audioPortLimit** is ignored if it is present

audioPortLimit

Type: **integer**

The limit on the number of audio ports this conference may allow.

This may be returned as **0**, even though the audio ports are not limited to **0**, unless **audioPortLimitSet** is **true**.

videoPortLimitSet

Type: **boolean**

Defines whether the **videoPortLimit** is applied.

Value	Description
true	Limits the number of video ports to the value in **videoPortLimit**
false	**videoPortLimit** is ignored if it is present

videoPortLimit

Type: **integer**

The limit on the number of video ports this conference may allow.

This may be returned as **0**, even though the video ports are not limited to **0**, unless **videoPortLimitSet** is **true**.

participantList

Type: **array**

Array of participants. Each member of the array is a struct that represents a participant on the TelePresence Server.

participantList array members

The returned **participantList** is an array of the conference's participants. Note that the member structs of this array are different to those returned in **participantList** by the **conference.invite**

call.

If there are no participants in this conference, the **participantList** array is returned empty.

Each struct contains the following parameters:

participantGUID

Type: **string**

The GUID of this participant, assigned by the TelePresence Server.

participantID

Type: **integer**

Deprecated. Use **participantGUID** instead.

The unique ID of this participant, assigned by the TelePresence Server.

The TelePresence Server returns this parameter, even though it is deprecated, to ensure application compatibility in the short term. We recommend that you use the replacement parameter instead.

callState

Type: **integer**

State of the call between the TelePresence Server and this participant.

Value	Description
0	Not connected
1	Calling in (not yet in conference)
2	Called in and participating
3	Calling out (not yet in conference)
4	Called out and participating

endpointType

Type: **integer**

Deprecated: use **endpointCategory** instead.

Value	Description
1	Normal endpoint
2	TANDBERG Experia
3	Grouped endpoints
4	T3
5	Cisco CTS or other TIP capable endpoints

The TelePresence Server returns this parameter, even though it is deprecated, to ensure application compatibility in the short term. We recommend that you use the replacement parameter instead.

endpointCategory

Type: **string**

Value	Description
normal	Normal endpoint
group	Grouped endpoints
exp	TANDBERG Experia
t3	T3
cts	Cisco CTS or other TIP capable endpoints

callStartMute

Type: **boolean**

True if this endpoint is being sent black video during call setup.

master

Type: **boolean**

Value	Description
true	This endpoint is conference master
false	(default if omitted) This endpoint is not the conference master

callType

Type: **string**

Value	Description
audio	An audio only participant
video	A video participant

callProtocol

Type: **string**

Value	Description
sip	This call uses the SIP protocol.
h323	This call uses the H.323 protocol.

Conditional:

enumerateID

Type: **string**

Enumerate calls may return many results so all of them will accept this parameter and may include this parameter in the response.

If the response includes an enumerateID, the application should pass the ID to the subsequent enumerate call to retrieve the next set of results. If the response does not include an enumerateID, there are no more results in the enumeration.

If the application omits the enumerateID, the target device will start a new enumeration and return the first set of results.

Only returned if there is more data to return than can be contained in one response.

disconnectReason

Type: **string**

The reason why the endpoint disconnected.

Value	Description
unspecified	Unspecified error
localTeardown	Requested by administrator
noAnswer	No answer
rejected	Call rejected
busy	Busy
gatekeeperError	Gatekeeper error
remoteTeardown	Left conference

Only returned for disconnected participants.

rxPreviewURL

Type: **string**

The URL to retrieve a jpeg snapshot of video received from this participant.

Only returned for active participants.

txPreviewURL

Type: **string**

The URL to retrieve a jpeg snapshot of video sent to this participant.

Only returned for active participants.

callDuration

Type: **integer**

The duration of the call in seconds.

Only returned for active participants.

callDirection

Type: **string**

This parameter is not present if `callState` is `0` (not connected).

Value	Description
incoming	The participant called in to the TelePresence Server
outgoing	The TelePresence Server called out to the participant

Only returned for active participants.

callBandwidth

Type: **integer**

Call bandwidth in kbps.

Only returned for active participants.

micMute

Type: **boolean**

True if far end microphone is muted.

recordingDevice

Type: **boolean**

Value	Description
true	The endpoint is treated as a recording device; it does not feature in the layout and other participants are made aware of its presence by a red dot as appropriate.
false	(default if omitted) The endpoint is a normal endpoint.

Only returned for active participants.

txAudioMute

Type: **boolean**

Defines whether the TelePresence Server mutes the audio signal transmitted to this endpoint.

Only returned for active participants.

rxAudioMute

Type: **boolean**

Defines whether the TelePresence Server mutes the audio signal received from this endpoint.

Only returned for active participants.

txVideoMute

Type: **boolean**

Defines whether the TelePresence Server mutes the video signal transmitted to this endpoint.

Only returned for active participants.

rxVideoMute

Type: **boolean**

Defines whether the TelePresence Server mutes the video signal received from this endpoint.

Only returned for active participants.

isImportant

Type: **boolean**

Defines whether the participant is important (i.e. the participant's transmitted video is given preference over others when composing video).

Value	Description
true	The participant is important
false	(Default if omitted) The participant's video is not given preference over other that of the other participants

Only returned for active participants.

conference.uninvite

Status: **active**

Removes participants from the specified conference. This call requires one conference identification parameter and one participant list parameter.

The call returns a fault if it cannot find a specified participant, even if the TelePresence Server has successfully uninvited the other specified participants.

Accepts:

Required:

To identify the conference, use **conferenceGUID** instead of **conferenceID**, not both.

conferenceGUID

Type: **string**

Globally unique identifier of the conference.

conferenceID

Type: **integer**

Deprecated. Use **conferenceGUID** instead.

Unique conference identifier.

Optional:

To identify participants to uninvite, **use only one of the following** optional parameters.

participantListGUID

Type: **string**

Comma separated list of **participantGUID**s that identifies which participants to remove from this conference. For example, **C8200C3F-49CE-4763-98E0-790B4F038995, B1101410-6BB8-487E-9D6F-91E810E80651**.

participantList

Type: **string**

A comma separated list of participant addresses.

Value	Description
Example string	**10.2.171.232, 10.47.2.246, h323:numericID@domain.com**

participantListID

Type: **string**

Deprecated. Use **participantListGUID** instead.

Comma separated list of **participantID**s that identifies which participants to remove from the conference. For example; **1024, 1056**.

participant.diagnostics

Status: **active**

The call specifies which participant's diagnostics to retrieve and also a listening interface for the returned information.

The reason for providing **receiverURI** is because the call is asynchronous; you should receive an "Operation successful" result slightly before the data returns (as an XML-RPC method called **participantDiagnosticsResponse**) on the listening interface.

The returned information contains arrays comprising the different types of data streams between this participant and the hosting TelePresence Server. Each array member represents a single stream.

Example XML-RPC response to participant.diagnostics

If there are no streams of a particular type, the corresponding array is returned empty.

Accepts:

Required:

participantGUID

Type: **string**

The GUID of this participant, assigned by the TelePresence Server.

receiverURI

Type: **string**

Fully-qualified URI that identifies the listening application's XML-RPC interface (protocol, address, and port), for example, `http://tms1:8080/RPC2`. You can use `http` or `https` and, if no port number is specified, the device will use the protocol defaults (80 and 443 respectively).

Optional:

sourceIdentifier

Type: **string**

The originating device uses this parameter to identify itself to the listening receiver/s. If the parameter is not explicitly set, the device identifies itself with the MAC address of its Ethernet port A interface.

Returns:

participantGUID

Type: **string**

The GUID of this participant, assigned by the TelePresence Server.

sourceIdentifier

Type: **string**

The originating device uses this parameter to identify itself to the listening receiver/s. If the parameter is not explicitly set, the device identifies itself with the MAC address of its Ethernet port A interface.

audioRx

Type: **array**

An array of structs, each of which represents an audio stream received from the participant's endpoint.

audioTx

Type: **array**

An array of structs, each of which represents an audio stream transmitted to the participant's endpoint.

auxiliaryAudioRx

Type: **array**

An array of structs, each of which represents an auxiliary audio stream received from the participant's endpoint.

auxiliaryAudioTx

Type: **array**

An array of structs, each of which represents an auxiliary audio stream transmitted to the participant's endpoint.

videoRx

Type: **array**

An array of structs, each of which represents a video stream received from the participant's endpoint.

videoTx

Type: **array**

An array of structs, each of which represents a video stream transmitted to the participant's endpoint.

contentVideoRx

Type: **array**

An array of structs, each of which represents a content video stream received from the participant's endpoint.

contentVideoTx

Type: **array**

An array of structs, each of which represents a content video stream transmitted to the participant's endpoint.

Contents of diagnostics arrays:

Each of the above arrays may contains zero or more stream structs. Each stream struct will contain relevant parameter/value pairs from the following lists:

All streams:

codec

Type: **string**

The codec in use, or **other** for undefined codecs.

encrypted

Type: **boolean**

True if the stream data is encrypted.

muted

Type: **boolean**

True if the stream is muted.

channelBitRate

Type: **integer**

Bit rate of the channel in bits per second (bps).

Conditional, depending on the type and direction of the stream:

packetsSent

Type: **integer**

Count of packets sent in this stream.

packetsReceived

Type: **integer**

Count of packets received in this stream.

packetErrors

Type: **integer**

Count of packets with errors in this stream.

packetsMissing

Type: **integer**

Count of packets missing from this stream.

framesReceived

Type: **integer**

Count of frames received in this stream.

frameErrors

Type: **integer**

Count of frames with errors in this stream.

jitter

Type: **integer**

Current jitter in this stream, measured in milliseconds (ms).

energy

Type: **integer**

The level of the signal, supplied in decibels (dB).

configuredBitRate

Type: **integer**

The configured bit rate of this stream, in bits per second (bps).

configuredBitRateReason

Type: **string**

Value	Description
aggregateBandwidth	The TelePresence Server has limited the bit rate so that multiple streams can be sent without exceeding a given limit on overall bandwidth.
flowControl	The far end has requested that the TelePresence Server sends video at a lower bit rate.
notLimited	The configured bit rate is not limited by **flowControl** or **aggregateBandwidth**.

expectedBitRate

Type: **integer**

The expected bit rate of this stream, in bits per second (bps).

expectedBitRateReason

Type: **string**

Value	Description
viewedSize	The TelePresence Server requested a reduction in the bitrate of the video stream because the video stream from that endpoint is not being displayed at full size.
errorPackets	The TelePresence Server requested a reduction in the bitrate of the video stream because there are errors in the video stream.
notLimited	The TelePresence Server has not requested a reduction in the bitrate of the video stream.

actualBitRate

Type: **integer**

The measured bit rate of this stream, in bits per second (bps).

frameRate

Type: **integer**

The frame rate of the video stream, in frames per second (fps).

fastUpdateRequestsSent

Type: **integer**

The count of fast update requests sent in this stream.

`fastUpdateRequestsReceived`

Type: **integer**

The count of fast update requests received in this stream.

participant.enumerate

Status: **active**

Returns an array of the participants who are active on the queried TelePresence Server. Endpoints that are either connecting or inactive at the time of the enumeration are not included in the response.

Accepts:

Optional:

`enumerateID`

Type: **string**

Enumerate calls may return many results so all of them will accept this parameter and may include this parameter in the response.

If the response includes an enumerateID, the application should pass the ID to the subsequent enumerate call to retrieve the next set of results. If the response does not include an enumerateID, there are no more results in the enumeration.

If the application omits the enumerateID, the target device will start a new enumeration and return the first set of results.

Returns:

If there are no participants to enumerate, then the **`participant.enumerate`** call does not return the **`participants`** array.

Conditional:

`participants`

Type: **array**

An array of structures that represent participants.

`enumerateID`

Type: **string**

Enumerate calls may return many results so all of them will accept this parameter and may include this parameter in the response.

If the response includes an enumerateID, the application should pass the ID to the subsequent enumerate call to retrieve the next set of results. If the response does not include an enumerateID, there are no more results in the enumeration.

If the application omits the enumerateID, the target device will start a new enumeration and return the first set of results.

`participants` array

`participantGUID`

Type: **string**

The GUID of this participant, assigned by the TelePresence Server.

`participantID`

Type: **integer**

Deprecated. Use **`participantGUID`** instead.

The unique ID of this participant, assigned by the TelePresence Server.

`conferenceGUID`

Type: **string**

Globally unique identifier of the conference.

`conferenceID`

Type: **integer**

Deprecated. Use **`conferenceGUID`** instead.

Unique conference identifier.

`address`

Type: **string**

The address of the item, e.g. endpoint or gateway; may be hostname, IP address or E.164 number.

`endpointCategory`

Type: **string**

Value	Description
normal	Normal endpoint
group	Grouped endpoints
exp	TANDBERG Experia
t3	T3
cts	Cisco CTS or other TIP capable endpoints

`callProtocol`

Type: **string**

Value	Description
sip	This call uses the SIP protocol.
h323	This call uses the H.323 protocol.

callState

Type: **integer**

State of the call between the TelePresence Server and this participant.

Value	Description
0	Not connected
1	Calling in (not yet in conference)
2	Called in and participating
3	Calling out (not yet in conference)
4	Called out and participating

participant.set

Status: **active**

Changes the state of the supplied parameters for the specified participant.

Accepts:

Required:

To identify the participant, use **participantGUID** instead of **participantID**, not both.

participantGUID

Type: **string**

The GUID of this participant, assigned by the TelePresence Server.

participantID

Type: **integer**

Deprecated. Use **participantGUID** instead.

The unique ID of this participant, assigned by the TelePresence Server.

Optional:

txAudioMute

Type: **boolean**

Defines whether the TelePresence Server mutes the audio signal transmitted to this endpoint.

rxAudioMute

Type: **boolean**

Defines whether the TelePresence Server mutes the audio signal received from this endpoint.

txVideoMute

Type: **boolean**

Defines whether the TelePresence Server mutes the video signal transmitted to this endpoint.

rxVideoMute

Type: **boolean**

Defines whether the TelePresence Server mutes the video signal received from this endpoint.

isImportant

Type: **boolean**

Defines whether the participant is important (i.e. the participant's transmitted video is given preference over others when composing video).

Value	Description
true	The participant is important
false	(Default if omitted) The participant's video is not given preference over other that of the other participants

participant.tidylayout

Status: **active**

Tidies up the composed video layout sent to the specified participant's endpoint.

Accepts:

Required:

participantGUID

Type: **string**

The GUID of this participant, assigned by the TelePresence Server.

participantID

Type: **integer**

Deprecated. Use `participantGUID` instead.

The unique ID of this participant, assigned by the TelePresence Server.

system.info

Status: **active**

Returns the current status of the queried system.

Returns:

gateKeeperOK

Type: **boolean**

True if the gatekeeper is configured and the TelePresence Server is registered.

makeCallsOK

Type: **boolean**

True if the system has enough resources to make at least one call.

tpsNumberOK

Type: **integer**

The count of configured and active TelePresence Servers.

tpdVersion

Type: **string**

The TelePresence Server software version number.

tpdName

Type: **string**

The TelePresence Server system name.

tpdUptime

Type: **integer**

The period of time (in seconds) that has passed since the system booted.

tpdSerial

Type: **string**

The serial number of the TelePresence Server.

portsVideoTotal

Type: **integer**

The total number of video ports.

portsVideoFree

Type: **integer**

Count of the currently unused video ports.

portsAudioTotal

Type: **integer**

The total number of audio ports.

portsAudioFree

Type: **integer**

Count of the currently unused audio ports.

portsContentTotal

Type: **integer**

The total number of content ports.

portsContentFree

Type: **integer**

Count of the currently unused content ports.

maxConferenceSizeVideo

Type: **integer**

The count of unused video ports on the least-used TelePresence Server controlled by this unit. Indicates the maximum number of video ports that could currently be allocated to a single conference.

maxConferenceSizeAudio

Type: **integer**

The count of unused audio-only ports on the least-used TelePresence Server controlled by this unit. Indicates the maximum number of audio-only ports that could currently be allocated to a single conference.

maxConferenceSizeContent

Type: **integer**

The count of unused content ports on the least-used TelePresence Server controlled by this unit. Indicates the maximum number of content ports that could currently be allocated to a single conference.

numControlledServers

Type: **integer**

The number of TelePresence Servers controlled by this unit (including itself).

Feedback receivers

The API allows you to register your application as a feedback receiver. This means that the application doesn't have to constantly poll the device if it wants to monitor activity.

The device publishes events when they occur. If the device knows that your application is listening for these events, it will send XML-RPC messages to your application's interface when the events occur.

- Use `feedbackReceiver.configure` to register a receiver to listen for one or more feedback events.
- Use `feedbackReceiver.query` to return a list of receivers that are configured on the device.
- Use `feedbackReceiver.reconfigure` to change the configuration of an existing feedback receiver.
- Use `feedbackReceiver.remove` to remove an existing feedback receiver.

After registering as a feedback receiver, the application will receive feedback messages on the specified interface.

feedbackReceiver.query

Status: **active**

This call asks the device for a list of all the feedback receivers that have previously been configured. It does not accept parameters other than the authentication strings.

Returns:

If there are no feedback receivers to enumerate, then the **`feedbackReceiver.query`** returns an empty **`receivers`** array.

`receivers`

Type: **array**

An array of feedback receivers, with members corresponding to the entries in the receivers table on the device's web interface.

`receivers` array members

Each receiver in the response contains the following parameters:

`receiverURI`

Type: **string**

Fully-qualified URI that identifies the listening application's XML-RPC interface (protocol, address, and port), for example, **`http://tms1:8080/RPC2`**. You can use **`http`** or **`https`** and, if no port number is specified, the device will use the protocol defaults (80 and 443 respectively).

`sourceIdentifier`

Type: **string**

The originating device uses this parameter to identify itself to the listening receiver/s. If the parameter is not explicitly set, the device identifies itself with the MAC address of its Ethernet port A interface.

index

Type: **integer**

A number that identifies the position of the item in context with similar items.

The **index** describes the position of this feedback receiver in the TelePresence Server's table of feedback receivers. It is a number between 1 and 20 (inclusive).

feedbackReceiver.configure

Status: **active**

This call configures the device to send feedback about the specified **subscribedEvents** to the specified **receiverURI**.

Accepts:

Required:

receiverURI

Type: **string**

Fully-qualified URI that identifies the listening application's XML-RPC interface (protocol, address, and port), for example, **http://tms1:8080/RPC2**. You can use **http** or **https** and, if no port number is specified, the device will use the protocol defaults (80 and 443 respectively).

Optional:

sourceIdentifier

Type: **string**

The originating device uses this parameter to identify itself to the listening receiver/s. If the parameter is not explicitly set, the device identifies itself with the MAC address of its Ethernet port A interface.

receiverIndex

Type: **integer**

The position of this feedback receiver in the device's table of feedback receivers.

Value	Description
-1	The feedback receiver will use any available position.
1	The first position in the table (this value is assumed if you don't supply **receiverIndex** - *overwriting any existing entry in position 1*)
20	The 20th (maximum allowed) position

We recommend that you set **`receiverIndex`** to **`-1`** in this call. This ensures that the TelePresence Server allocates an available slot and that you don't inadvertently overwrite an existing feedback receiver in slot 1.

`subscribedEvents`

Type: **array**

An array of strings, each of which is the name of a notification event. The array defines the events to which the receiver subscribes.

You may specify any or all of the following:

- cdrAdded
- conferenceStarted
- conferenceFinished
- conferenceActive
- conferenceInactive
- configureAck
- participantJoined
- participantLeft
- participantConnected
- participantDisconnected
- restart

If this array is absent, the receiver subscribes to all notifications by default.

Returns:

The call returns the allocated **`receiverIndex`**.

`receiverIndex`

Type: **integer**

The position of this feedback receiver in the device's table of feedback receivers.

feedbackReceiver.reconfigure

Status: **active**

This call reconfigures an existing feedback receiver.This call only reconfigures the receiver parameters that you specify; the TelePresence Server retains the original values for any parameters that you omit.

Accepts:

Required:

`receiverIndex`

Type: **integer**

The position of this feedback receiver in the device's table of feedback receivers.

The call returns a fault if there is no feedback receiver at the specified **`receiverIndex`**.

Optional:

`receiverURI`

Type: **string**

Fully-qualified URI that identifies the listening application's XML-RPC interface (protocol, address, and port), for example, **`http://tms1:8080/RPC2`**. You can use **`http`** or **`https`** and, if no port number is specified, the device will use the protocol defaults (80 and 443 respectively).

The call returns a fault if you supply an empty **`receiverURI`**. However, if you omit the parameter altogether, the original value persists.

`sourceIdentifier`

Type: **string**

The originating device uses this parameter to identify itself to the listening receiver/s. If the parameter is not explicitly set, the device identifies itself with the MAC address of its Ethernet port A interface.

`subscribedEvents`

Type: **array**

An array of strings, each of which is the name of a notification event. The array defines the events to which the receiver subscribes.

You may specify any or all of the following:

- cdrAdded
- conferenceStarted
- conferenceFinished
- conferenceActive
- conferenceInactive
- configureAck
- participantJoined
- participantLeft
- participantConnected
- participantDisconnected
- restart

If this array is absent, the receiver's existing subscriptions will not be changed from the values created by the original **`feedbackReceiver.configure`** call.

feedbackReceiver.remove

Status: **active**

Removes the specified feedback receiver.

Accepts:

Required:

`receiverIndex`

Type: **integer**

The position of this feedback receiver in the device's table of feedback receivers.

The call returns a fault if there is no feedback receiver at the specified **`receiverIndex`**.

Feedback messages

The feedback messages follow the format used by the device for XML-RPC responses.

The messages contain two parameters:

- **`sourceIdentifier`** is a string that identifies the device, which may have been set by **`feedbackReceiver.configure`** or otherwise will be the device's MAC address.
- **`events`** is an array of strings that contain the names of the feedback events that have occurred.

Example feedback message

```
<?xml version="1.0" encoding="UTF-8" ?>
<methodCall>
  <methodName>eventNotification</methodName>
  <params>
    <param>
      <value>
        <struct>
          <member>
            <name>sourceIdentifier</name>
            <value><string>000D7C000C66</string></value>
          </member>
          <member>
            <name>events</name>
            <value>
              <array>
                <data>
                  <value><string>restart</string></value>
                </data>
              </array>
            </value>
          </member>
        </struct>
      </value>
    </param>
  </params>
</methodCall>
```

Feedback events

The following table lists the feedback events that the TelePresence Server can publish.

Event	Description
restart	The source publishes this event when it starts up
configureAck	The source publishes this event to acknowledge that an application has successfully configured a feedback receiver
cdrAdded	One or more new Call Detail Records have been logged
conferenceStarted	One or more conferences have been created
conferenceFinished	One or more conferences have been deleted
conferenceActive	One or more conferences have become active (first participant joined)
conferenceInactive	One or more conferences have become inactive (last participant left)
participantJoined	One or more participants have joined a conference
participantLeft	One or more participants have left a conference
participantConnected	One or more participants have connected to the TelePresence Server
participantDisconnected	One or more participants disconnected from the TelePresence Server

Related information

system.xml file

You can derive some information about the TelePresence Server from its **system.xml** file. You can download this file via HTTP from the TelePresence Server's root.

Example **system.xml**

```
<?xml version="1.0"?>
  <system>
    <manufacturer>TANDBERG</manufacturer>
    <model>Telepresence Server 8710</model>
    <serial>SM0215D3</serial>
    <softwareVersion>2.2(1.43)</softwareVersion>
    <buildVersion>9.5(1.43)</buildVersion>
    <hostName/>
    <ipAddress>10.47.223.38</ipAddress>
    <ipAddressV6>2001:420:40ff:ff0a:20d:7cff:fe10:ae98</ipAddressV6>
    <macAddress>00:0D:7C:10:AE:98</macAddress>
    <gatekeeperUsage>Yes</gatekeeperUsage>
    <gatekeeperAddress>10.47.212.105</gatekeeperAddress>
    <gatekeeperIds>dt3b8-1000,dt3b8-1000-1,dt3b8-1000-c,dt3b8-1000-r</gatekeeperIds>
    <sipRegistrarUsage>Yes</sipRegistrarUsage>
    <sipRegistrarAddress>10.47.212.105</sipRegistrarAddress>
    <sipRegistrarDomain>vcs5.test.lal</sipRegistrarDomain>
    <sipTrunkUsage>No</sipTrunkUsage>
    <sipTrunkAddress/>
    <sipTrunkDomain/>
    <isMaster>Yes</isMaster>
    <clusterType>master</clusterType>
    <totalVideoPorts>48</totalVideoPorts>
    <totalContentPorts>40</totalContentPorts>
    <totalAudioOnlyPorts>40</totalAudioOnlyPorts>
    <uptimeSeconds>532236</uptimeSeconds>
  </system>
```

System XML contents

Node name	Node contents
gatekeeperUsage	**Yes**: gatekeeper usage is enabled **No**: gatekeeper usage is disabled
gatekeeperIds	Comma separated list of registered IDs associated with this TelePresence Server and its slaves (omitted if the system is not a master)
isMaster	**Yes**: this system is a master **No**: this system is a slave
sipRegistrarUsage	**Yes**: registrar usage is enabled **No**: registrar usage is disabled

Node name	Node contents
sipTrunkUsage	**Yes**: trunk usage is enabled **No**: trunk usage is disabled
clusterType	The role of this system in a backplane cluster. May be **unclustered**, **master**, or **slave**

Example XML-RPC response to `participant.diagnostics`

```
<?xml version="1.0" encoding="UTF-8" ?>
<methodCall>
  <methodName>participantDiagnosticsResponse</methodName>
  <params>
    <param>
      <value>
        <struct>
          <member>
            <name>participantGUID</name>
            <value>
              <string>1f983510-8843-11e0-bd5d-000d7c005ce8</string>
            </value>
          </member>
          <member>
            <name>sourceIdentifier</name>
            <value>
              <string>UserPC</string>
            </value>
          </member>
          <member>
            <name>audioRx</name>
            <value>
              <array>
                <data>
                  <value>
                    <struct>
                      <name>codec</name>
                      <value>
                        <string>G.722</string>
                      </value>
                      <name>encrypted</name>
                      <value>
                        <boolean>1</boolean>
                      </value>
                      <name>channelBitRate</name>
                      <value>
                        <int>64000</int>
                      </value>
                      <name>jitter</name>
                      <value>
                        <int>0</int>
                      </value>
                      <name>energy</name>
                      <value>
                        <int>-34</int>
```

```
                </value>
                <name>packetsReceived</name>
                <value>
                  <int>6951</int>
                </value>
                <name>packetErrors</name>
                <value>
                  <int>0</int>
                </value>
                <name>packetsMissing</name>
                <value>
                  <int>0</int>
                </value>
                <name>framesReceived</name>
                <value>
                  <int>277960</int>
                </value>
                <name>frameErrors</name>
                <value>
                  <int>0</int>
                </value>
                <name>muted</name>
                <value>
                  <boolean>0</boolean>
                </value>
              </struct>
            </value>
          </data>
        </array>
      </value>
    </member>
    <member>
      <name>audioTx</name>
      <value>
        <array>
          <data>
            <value>
              <struct>
                <name>codec</name>
                <value>
                  <string>G.722</string>
                </value>
                <name>encrypted</name>
                <value>
                  <boolean>1</boolean>
                </value>
                <name>channelBitRate</name>
                <value>
                  <int>64000</int>
                </value>
                <name>packetsSent</name>
                <value>
                  <int>6994</int>
                </value>
                <name>muted</name>
                <value>
                  <boolean>0</boolean>
```

```
            </value>
          </struct>
        </value>
      </data>
    </array>
  </value>
</member>
<member>
  <name>auxiliaryAudioRx</name>
  <value>
    <array>
      <data/>
    </array>
  </value>
</member>
<member>
  <name>auxiliaryAudioTx</name>
  <value>
    <array>
      <data/>
    </array>
  </value>
</member>
<member>
  <name>videoRx</name>
  <value>
    <array>
      <data>
        <value>
          <struct>
            <name>codec</name>
            <value>
              <string>H.264</string>
            </value>
            <name>height</name>
            <value>
              <int>0</int>
            </value>
            <name>width</name>
            <value>
              <int>0</int>
            </value>
            <name>encrypted</name>
            <value>
              <boolean>1</boolean>
            </value>
            <name>channelBitRate</name>
            <value>
              <int>448000</int>
            </value>
            <name>expectedBitRate</name>
            <value>
              <int>256000</int>
            </value>
            <name>expectedBitRateReason</name>
            <value>
              <string>viewedSize</string>
```

```
            </value>
            <name>actualBitRate</name>
            <value>
              <int>0</int>
            </value>
            <name>jitter</name>
            <value>
              <int>0</int>
            </value>
            <name>packetsReceived</name>
            <value>
              <int>0</int>
            </value>
            <name>packetErrors</name>
            <value>
              <int>0</int>
            </value>
            <name>framesReceived</name>
            <value>
              <int>0</int>
            </value>
            <name>frameErrors</name>
            <value>
              <int>0</int>
            </value>
            <name>frameRate</name>
            <value>
              <int>0</int>
            </value>
            <name>fastUpdateRequestsSent</name>
            <value>
              <int>0</int>
            </value>
            <name>muted</name>
            <value>
              <boolean>0</boolean>
            </value>
          </struct>
        </value>
      </data>
    </array>
  </value>
</member>
<member>
  <name>videoTx</name>
  <value>
    <array>
      <data>
        <value>
          <struct>
            <name>codec</name>
            <value>
              <string>H.263+</string>
            </value>
            <name>height</name>
            <value>
              <int>576</int>
```

```
                </value>
                <name>width</name>
                <value>
                  <int>704</int>
                </value>
                <name>encrypted</name>
                <value>
                  <boolean>1</boolean>
                </value>
                <name>channelBitRate</name>
                <value>
                  <int>448000</int>
                </value>
                <name>configuredBitRate</name>
                <value>
                  <int>448000</int>
                </value>
                <name>configuredBitRateReason</name>
                <value>
                  <string>notLimited</string>
                </value>
                <name>actualBitRate</name>
                <value>
                  <int>47468</int>
                </value>
                <name>packetsSent</name>
                <value>
                  <int>2054</int>
                </value>
                <name>frameRate</name>
                <value>
                  <int>15</int>
                </value>
                <name>fastUpdateRequestsReceived</name>
                <value>
                  <int>2</int>
                </value>
                <name>muted</name>
                <value>
                  <boolean>0</boolean>
                </value>
              </struct>
            </value>
          </data>
        </array>
      </value>
    </member>
    <member>
      <name>contentVideoRx</name>
      <value>
        <array>
          <data/>
        </array>
      </value>
    </member>
    <member>
      <name>contentVideoTx</name>
```

```
            <value>
              <array>
                <data/>
              </array>
            </value>
          </member>
        </struct>
      </value>
    </param>
  </params>
</methodCall>
```

Fault codes

The Cisco TelePresence Server returns a fault code when it encounters a problem with processing an XML-RPC request.

The following table lists the fault codes that may be returned by the TelePresence Server and their most common interpretations.

Fault Code	Description
2	**`Duplicate conference name`**. A conference name was specified, but is already in use.
4	**`No such conference or auto attendant`**. The conference or auto attendant identification given does not match any conference or auto attendant.
5	**`No such participant`**. The participant identification given does not match any participants.
6	**`Too many conferences`**. The device has reached the limit of the number of conferences that can be configured.
8	**`No conference name or auto attendant id supplied`**. A conference name or auto attendant identifier was required, but was not present.
10	**`No participant address supplied`**. A participant address is required but was not present.
15	**`Insufficient privileges`**. The specified user id and password combination is not valid for the attempted operation.
16	**`Invalid enumerateID value`**. An enumerate ID passed to an enumerate method invocation was invalid. Only values returned by the device should be used in enumerate methods.
17	**`Port reservation failure`**. This is in the case that reservedAudioPorts or reservedVideoPorts value is set too high, and the device cannot support this.
18	**`Duplicate numeric ID`**. A numeric ID was given, but this ID is already in use.
20	**`Unsupported participant type`**. A participant type was used which does not correspond to any participant type known to the device.
25	**`New port limit lower than currently active`**
35	**`String is too long`**. The call supplied a string parameter that was longer than allowed. Strings are usually limited to 32 characters but may be 63 in some cases.
101	**`Missing parameter`**. This is given when a required parameter is absent. The parameter in question is given in the fault string in the format "missing parameter - parameter_name".

102	**`Invalid parameter`**. This is given when a parameter was successfully parsed, is of the correct type, but falls outside the valid values; for example an integer is too high or a string value for a protocol contains an invalid protocol. The parameter in question is given in the fault string in the format "invalid parameter - parameter_name".
103	**`Malformed parameter`**. This is given when a parameter of the correct name is present, but cannot be read for some reason; for example the parameter is supposed to be an integer, but is given as a string. The parameter in question is given in the fault string in the format "malformed parameter - parameter_name".
105	**`Request too large`**. The method call contains more data than the API can accept. The maximum size of the call is 32 kilobytes.
201	**`Operation failed`**. This is a generic fault for when an operation does not succeed as required.

HTTP keep-alives

Your application can use use HTTP keep-alives to reduce the amount of TCP traffic that results from constantly polling the device. Any client which supports HTTP keep-alives may include the following line in the HTTP header of an API request:

```
Connection: Keep-Alive
```

This indicates to the device that the client supports HTTP keep-alives. The device may then choose to maintain the TCP connection after it has responded. If the device will close the connection it returns the following HTTP header in its response:

```
Connection: close
```

If this line is not in the HTTP header of the response, the client may use the same connection for a subsequent request.

The device will not keep a connection alive if:

- the current connection has already serviced the allowed number of requests
- the current connection has already been open for the allowed amount of time
- the number of open connections exceeds the allowed number if this connection is maintained

These restrictions are in place to limit the resources associated with open connections. If a connection is terminated for either of the first two reasons, the client will probably find that the connection is maintained after the next request.

Note: The client should never assume a connection will be maintained. Also, the device will close an open connection if the client does not make any further requests within a minute. There is little benefit to keeping unused connections open for such long periods.

Checking for updates and getting help

If you experience any problems when configuring or using the product, consult the online help available from the user interface. The online help explains how the individual features and settings work.

If you cannot find the answer you need, check the web site at http://www.cisco.com/en/US/products/ps11339/tsd_products_support_series_home.html where you will be able to:

- make sure that you are running the most up-to-date software,
- find further relevant documentation, for example product user guides, printable versions of the online help, reference guides, and articles that cover many frequently asked questions,
- get help from the Cisco Technical Support team. Make sure you have the following information ready before raising a case:
  - the serial number and product model number of the unit (if applicable)
  - the software build number which can be found on the product user interface (if applicable)
  - your contact email address or telephone number
  - a full description of the problem

References

1. XML-RPC specification (Dave Winer, June 1999); http://www.xmlrpc.com/spec, accessed 24/01/2011.
2. HTTP/1.1 specification (RFC 2616, Fielding et al., June 1999); http://www.ietf.org/rfc/rfc2616.txt, accessed 24/01/2011.
3. Cisco TelePresence CDR logs reference guide (Cisco Systems Inc., June 2011); http://www.cisco.com/en/US/docs/telepresence/infrastructure/mcu/admin_guide/Cisco_TelePresence_infrastructure_products_CDR_log_reference.pdf, accessed 07/07/2011.

THE SPECIFICATIONS AND INFORMATION REGARDING THE PRODUCTS IN THIS MANUAL ARE SUBJECT TO CHANGE WITHOUT NOTICE. ALL STATEMENTS, INFORMATION, AND RECOMMENDATIONS IN THIS MANUAL ARE BELIEVED TO BE ACCURATE BUT ARE PRESENTED WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. USERS MUST TAKE FULL RESPONSIBILITY FOR THEIR APPLICATION OF ANY PRODUCTS.

THE SOFTWARE LICENSE AND LIMITED WARRANTY FOR THE ACCOMPANYING PRODUCT ARE SET FORTH IN THE INFORMATION PACKET THAT SHIPPED WITH THE PRODUCT AND ARE INCORPORATED HEREIN BY THIS REFERENCE. IF YOU ARE UNABLE TO LOCATE THE SOFTWARE LICENSE OR LIMITED WARRANTY, CONTACT YOUR CISCO REPRESENTATIVE FOR A COPY.

The Cisco implementation of TCP header compression is an adaptation of a program developed by the University of California, Berkeley (UCB) as part of UCB's public domain version of the UNIX operating system. All rights reserved. Copyright © 1981, Regents of the University of California.

NOTWITHSTANDING ANY OTHER WARRANTY HEREIN, ALL DOCUMENT FILES AND SOFTWARE OF THESE SUPPLIERS ARE PROVIDED "AS IS" WITH ALL FAULTS. CISCO AND THE ABOVE-NAMED SUPPLIERS DISCLAIM ALL WARRANTIES, EXPRESSED OR IMPLIED, INCLUDING, WITHOUT LIMITATION, THOSE OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OR ARISING FROM A COURSE OF DEALING, USAGE, OR TRADE PRACTICE.

IN NO EVENT SHALL CISCO OR ITS SUPPLIERS BE LIABLE FOR ANY INDIRECT, SPECIAL, CONSEQUENTIAL, OR INCIDENTAL DAMAGES, INCLUDING, WITHOUT LIMITATION, LOST PROFITS OR LOSS OR DAMAGE TO DATA ARISING OUT OF THE USE OR INABILITY TO USE THIS MANUAL, EVEN IF CISCO OR ITS SUPPLIERS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Cisco and the Cisco Logo are trademarks of Cisco Systems, Inc. and/or its affiliates in the U.S. and other countries. A listing of Cisco's trademarks can be found at www.cisco.com/go/trademarks. Third party trademarks mentioned are the property of their respective owners. The use of the word partner does not imply a partnership relationship between Cisco and any other company. (1005R)

Any Internet Protocol (IP) addresses and phone numbers used in this document are not intended to be actual addresses and phone numbers. Any examples, command display output, network topology diagrams, and other figures included in the document are shown for illustrative purposes only. Any use of actual IP addresses or phone numbers in illustrative content is unintentional and coincidental.