Cisco TelePresence TelePresence Server API

Product Programming Reference Guide

4.0(2.8)

D15049.01

June 2014

Overview

This guide describes the APIs available in version 4.0(2.8) of Cisco TelePresence Server:

- Part 1: Flexible operation mode [p.3] describes the API available when the operation mode is set to **`flexible`**. This corresponds to the *remotely managed* mode of operation as described in the user interface and online help.
- Part 2: Standalone operation mode [p.107] describes the API available when the operation mode is set to **`standalone`**. This corresponds to the *locally managed* mode of operation as described in the user interface and online help.

The **`operationMode`** parameter of the **`system.info`** method returns the current operation mode.

Part 1: Flexible operation mode

Part 1 of this guide describes the API available in flexible operation mode (remotely managed). For information about the API available in standalone operation mode (locally managed), refer to Part 2: Standalone operation mode [p.107].

Introduction ... 5
- Remotely managed mode API change summary ... 5
- Design considerations ... 5

XML-RPC implementation ... 7
- Transport ... 7
- Encoding ... 7
- Message flow ... 8
- Data types and sizes ... 9
- HTTP keep-alives ... 10

API overview ... 11
- Authentication ... 11
- Identifiers and client references ... 11
- Conference URI identifiers ... 12
- Participants ... 13
- "Unlimited" integers ... 15
- Media credits ... 15
- Media reservation ... 16
- Enumeration ... 16
- DTMF ... 19
- Feedback receivers ... 19

Data structures and enumerated types ... 22
- Enumerated types ... 22
- Structs ... 27

API command reference ... 36
- callHome.configure ... 37
- callHome.query ... 37
- cdrlog.enumerate ... 38
- cdrlog.query ... 39
- device.feature.add ... 40
- device.feature.remove ... 40
- device.health.query ... 40
- device.network.query ... 41
- device.query ... 43
- device.restart ... 44
- device.restartlog.query ... 44
- feedbackReceiver.configure ... 45
- feedbackReceiver.query ... 46
- feedbackReceiver.reconfigure ... 47
- feedbackReceiver.remove ... 47
- feedbackReceiver.status ... 47
- flex.call.status ... 48
- flex.conference.create ... 49
- flex.conference.deletions.enumerate ... 53

flex.conference.destroy 54
flex.conference.enumerate 54
flex.conference.getMetadata 56
flex.conference.modify 57
flex.conference.query 61
flex.conference.sendUserMessage 64
flex.conference.sendWarning 65
flex.conference.status 65
flex.participant.advanced.enumerate 66
flex.participant.call.disconnect 69
flex.participant.clearImportant 70
flex.participant.create 70
flex.participant.deletions.enumerate 72
flex.participant.destroy 73
flex.participant.enumerate 73
flex.participant.media.enumerate 75
flex.participant.modify 77
flex.participant.query 77
flex.participant.requestDiagnostics 79
flex.participant.requestPreview 85
flex.participant.sendDTMF 87
flex.participant.sendUserMessage 88
flex.participant.setImportant 88
flex.participant.setMute 88
flex.participant.status 89
flex.resource.query 91
flex.resource.status 93
system.info 93
Related information 96
system.xml on 8710 and 7010 96
system.xml on Media 310/320 98
system.xml on Virtual Machine 99
Fault codes 100
Example XML-RPC response to flex.conference.create 101
Remotely managed API change history 103

Introduction

This document accompanies the latest version of the management API for the Cisco TelePresence Server software when running in flexible (remotely managed) mode. The following Cisco TelePresence products support this API when they are running TelePresence Server version 4.0(2.8) and later:

- Cisco TelePresence Server MSE 8710
- Cisco TelePresence Server 7010
- Cisco TelePresence Server on Multiparty Media 310/320
- Cisco TelePresence Server on Virtual Machine

Remotely managed mode API change summary

The latest Cisco TelePresence Server API is version 4.0(2.8). The table below contains a summary of the latest changes to the remotely managed mode API. For changes introduced in older versions, see Remotely managed API change history [p.103].

Table 1: API version 4.0(2.8) change summary

XML-RPC Request / Topic	Parameter	Change
callHome.configure [p.37]	**`mode`**, **`automatic`**	New command
callHome.query [p.37]	**`mode`**, **`automatic`**	New command
system.info [p.158]	**`cpuModel`** **`cpuCount`** **`cpuAvx`**	Added

Design considerations

Every API command that your application sends incurs a processing overhead within the device’s own application. The amount of the overhead varies widely with the type of command and the parameters sent. If the device receives a high number of API commands every second, its performance could be seriously impaired (in the same way as if multiple users simultaneously accessed it via the web interface).

Minimizing API overhead

It is essential to design your application architecture and software so that the processing load on the device application is minimized.

To do this we recommend that you do the following:

- Use a single server to run the API application and to send 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. Then use the server to 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 requests.

These measures provide much more control than having the clients send API commands directly, and will prevent the device performance being impaired by unmanageable numbers of API requests.

Unavailable or irrelevant data

The API is designed to minimize impact on the network when responding to requests, and device responses do not routinely include either irrelevant data or empty data structures where the data is unavailable.

It follows that your application should take responsibility for checking whether a response includes the expected data, and should be designed for graceful handling of situations where the device does not respond with the expected data.

XML-RPC implementation

The API is implemented as messages sent using the XML-RPC protocol. This is a simple protocol for remote procedure calling that uses HTTP (or HTTPS) as the transport and XML as the encoding. XML-RPC is designed to be as simple as possible while allowing for complex data structures to be transmitted, processed and returned. It has no platform or software dependence and was chosen in favor of SOAP (Simple Object Access Protocol) because of its simplicity.

The API implements all parameters and returned data as **`<struct>`** elements, each of which is explicitly named. For example, the device.query call returns the current time as a structure member named currentTime rather than as a single <dateTime.iso8601> value:

```
<member>
  <name>
    currentTime
  </name>
  <value>
    <dateTime.iso8601>
      20130218T10:45:00
    </dateTime.iso8601>
  </value>
</member>
```

Refer to the XML-RPC specification for more information.

Transport

The device implements HTTP/1.1 as defined by RFC 2616. 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.

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"?>`**.

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>flex.conference.destroy</methodName>
    <params>
      <param>
        <value>
          <struct>
            <member>
              <name>authenticationPassword</name>
              <value><string></string></value>
            </member>
            <member>
              <name>conferenceID</name>
              <value><string>6f030fa0-08c4-11e2-a57e-000d07100000</string></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:

- If the command was successful:
  - If the API method returns parameters, the device responds with an XML <methodResponse> message containing a structure of return parameters, as documented for each command in the API command reference [p.36].
  - If the API method does not return parameters, the device responds with an XML <methodResponse> message containing a structure consisting of the single element `status` with value `operation successful`.
- If the command was unsuccessful, the device responds with an XML <methodResponse> that includes only a <fault> structure. See Fault codes [p.100].

Example success response where the API method does not return parameters

```
<?xml version="1.0" encoding="UTF-8"?>
  <methodResponse>
    <params>
      <param>
        <value>
          <struct>
```

```
            <member>
              <name>status</name>
              <value>
                <string>operation successful</string>
              </value>
            </member>
          </struct>
        </value>
      </param>
    </params>
  </methodResponse>
```

Example fault response

```
<?xml version="1.0" encoding="UTF-8"?>
  <methodResponse>
    <fault>
      <value>
        <struct>
          <member>
            <name>faultCode</name>
            <value>
              <int>4</int>
            </value>
          </member>
          <member>
            <name>faultString</name>
            <value>
              <string>conferenceID: no such conference</string>
            </value>
          </member>
        </struct>
      </value>
    </fault>
  </methodResponse>
```

Data types and sizes

Note: The total size of a request or response is 32 kB. If the TelePresence Server needs to truncate a response it will either provide a mechanism for you to retrieve the remaining data or return an appropriate fault code.

The Cisco TelePresence Server API accepts the following XML-RPC types. The table includes the default sizes that your application can assume unless a more specific limit is given in a parameter description.

Table 2: API data types and sizes

Type	Default size accepted
<string>	31 characters
<int>	Four byte signed (-2147483648 to 2147483647)
<boolean>	`1` or `0`, **`true`** or **`false`**
<base64>	Not explicitly limited unless otherwise stated

Table 2: API data types and sizes (continued)

Type	Default size accepted
<dateTime.iso8601>	ISO 8601 format eg. `20140107T13:31:26`
<array>	N/A
<struct>	N/A

HTTP keep-alives

Your application can 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.

API overview

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 the following authentication parameters:

Table 3: Authentication parameters

Parameter name	Type	Description
`authenticationUser`	string	**Required**. User name.
`authenticationPassword`	string	**Required**. User password.

If the user name and password are not recognized by the TelePresence Server, the method call fails with authentication errors.

Identifiers and client references

Identifiers and client references are string fields up to 50 characters in length.

Identifiers

The TelePresence Server assigns identifiers to resources and conferencing objects (conferences, calls). Identifiers within a pool of resource or object types are unique.

API clients must use identifiers to refer to resources and conferencing objects. The format and content of the identifier strings is subject to change and clients should not rely on any characteristics of identifiers.

Identifier fields have well-defined names that are used consistently within the TelePresence Server XML-RPC schema:

- **`conferenceID`**: unique identifier for a conference assigned by the TelePresence Server at conference instantiation.
- **`callID`**: unique identifier for a call assigned by the TelePresence Server at call instantiation.
- **`participantID`**: unique identifier for a participant. A participant can have one or more associated calls.

Client references

Client references are strings associated with objects created by this API. The content of these strings is set by clients of this API.

Client references make it possible for clients to create their own associations for objects.

The TelePresence Server does not use client references for any purpose other than to return the client reference associated with an object on request.

Client reference fields have well-defined names that are used consistently within the XML-RPC schema of this API:

- **`conferenceReference`**: client reference for conferences.
- **`participantReference`**: client reference for participants.

Client reference strings are only returned if they are not empty.

Conference URI identifiers

The conference URI is an identifier that allows matching of incoming calls to conferences. A conference URI can take either of the following forms:

- username@domain
- 123-ABC_example.com

Valid characters are as follows:

- 0 through 9
- a through z
- A through Z
- .-_@ (only one occurrence of @ is allowed)

URI matching and connection of incoming calls

Suppose that incoming calls dial in to an address on a TelePresence Server. The address dialed by the incoming call is matched to a URI and connected to a conference using the following algorithm:

1. Search for a URI that is an exact match for the address. If found, connect the call to the associated conference.
2. Strip the domain part of the address (if any) and search for a URI that is an exact match. If found, connect the call to the associated conference.
3. Reject the call.

Examples of URI matching

These examples illustrate how matching works for conference URIs with domains.

1. URI = **`conference_1@example.com`**
   - Call to **`conference_1@example.com`** will succeed
   - Call to **`conference_1@tower.example.com`** will fail
   - Call to **`conference_1`** will fail
2. URI = **`123456@example.com`**
   - Call to **`123456@example.com`** will succeed
   - Call to **`123456@tower.example.com`** will fail
   - Call to **`123456`** will fail
3. URI = **`conference_1`**
   - Call to **`conference_1@example.com`** will succeed
   - Call to **`conference_1@tower.example.com`** will succeed
   - Call to **`conference_1`** will succeed
4. URI = **`123456`**

- Call to `123456@example.com` will succeed
- Call to `123456@tower.example.com` will succeed
- Call to `123456` will succeed

5. Conference 1 has URI = `789`. Conference 2 has URI = `789@tower.example.com`. Conference 3 has URI = `789@example.com`
   - Call to `789@example.com` will succeed in being connected to conference 3.
   - Call to `789@tower.example.com` will succeed in being connected to conference 2.
   - Call to `789` will succeed in being connected to conference 1.
6. Conference 1 has URI = `789`. Conference 2 has URI = `789@example.com`
   - Call to `789@example.com` will succeed in being connected to conference 2.
   - Call to `789@tower.example.com` will succeed in being connected to conference 1.
   - Call to `789` will succeed in being connected to conference 1.

Participants

A participant can be an entity connected to a conference using one or more calls. Participant connections can be any of the following:

- Single-screen, single call connection.
- Multiscreen, single call connection.
- Multiscreen, multiple call connection.

Participants are implicitly created for incoming calls connecting to conference URIs. All other participants must be explicitly created using the API.

The API supports the creation of single and multi-call participants for which the calls can be incoming or outgoing. In the case of multi-call participants, the API supports combinations of incoming and outgoing calls.

Participant conference URIs

A participant can have associated conference URIs that are distinct from the URIs defined for a conference. These are called participant conference URIs. Each participant conference URI supports a single active call only. Incoming calls on participant conference URIs are connected to the conference as defined by the participant.

Participant conference URIs are bound to the conference and hence the activation and lifetimes do not exceed conference activation and lifetimes.

A participant can be configured to allow further incoming calls on a participant conference URI to be rejected or to replace the existing call.

Creating outgoing calls

The following rules apply for participant outgoing call creation:

- If all the participant calls are outgoing, the calls are created immediately (that is, on creation of the participant).
- If some but not all of the calls are outgoing, the outgoing calls are created after all incoming calls for the participant have connected.

- A PIN is not accepted by flex.participant.create [p.70] if all the calls are outgoing, because the TelePresence Server never requests a PIN when it has dialed out to an endpoint; in this case, the TelePresence Server will return fault code 102.

Participant attributes

Each participant has a unique identifier (assigned by the TelePresence Server) : **`participantID`**, and optionally a client-supplied reference: **`participantReference`**. See Identifiers and client references [p.11].

After a participant has been created, only the display name, call attributes, and media resources can be modified.

A single set of call attributes is defined for a participant, which apply to all calls belonging to a participant.

Each participant has a call nominated as the content transmitter and receiver and another as the audio transmitter and receiver. In the case of single-call participants, the content and audio transmitter and receiver can only be the single call that forms the participant.

A single PIN number specification is used and this can be input on the call nominated as the audio transmitter and receiver.

The media credits and tokens configured for a participant are reserved by the TelePresence Server for use by the calls that are members of the participant. The reservation exists for the lifetime of the participant.

All methods except for flex.participant.create [p.70] require the **`participantID`** field to identify a participant in the conference. If the **`participantID`** supplied is invalid, methods fail with a "no such participant" fault.

All methods that return information return the **`participantID`** field, and the client-supplied **`participantReference`** field if one was supplied.

Participant lifespan

A participant is associated with one and only one conference. The lifetime of a participant cannot exceed the lifetime of the conference with which it is associated. The activation time of a participant is bound to the activation time of the conference.

A participant is destroyed automatically when any call belonging to the participant hangs up. The exception to this rule applies to participants created using this API that have incoming calls: these participants persist for the duration of the conference, unless they are destroyed explicitly using this API. Also, if any one call belonging to such a participant hangs up, all other calls connected to the participant are disconnected.

If a participant is configured with **`deferConnect`** enabled, then it is not destroyed when its calls are disconnected. The participant remains in the conference and will be redialed when other participants join.

Participant media distribution

The media resource values are distributed to calls forming the participant according to the following rules:

- Main video tokens are divided appropriately amongst all calls in the participant.
- Extended video tokens are assigned to the nominated content transmitter and receiver.
- Audio tokens are assigned to the nominated audio transmitter and receiver.
- Media credits must be sufficient for the sum of all tokens specified.

"Unlimited" integers

Some parameters exchanged by this API represent configuration options that can have a greater value than what can be represented by a four byte integer.

For these options, the API and the client application can exchange a boolean version of the integer parameter which is set to **`true`** if the integer is unlimited. The naming convention for the boolean parameter is to append **`Unlimited`** to the name of the associated integer parameter.

When such a value is exchanged, only one of the two types may be supplied and only one will be returned. The Cisco TelePresence Server adheres strictly to this rule, and will return a fault if your application attempts to pass both.

For example, consider an integer field called **`duration`** with valid values >= 0. The associated boolean field is named **`durationUnlimited`**.

The following table describes the XML encoding for all settings of **`duration`**.

Table 4: Example of "Unlimited" integer

Value	XML		
	name	type	value
0 to 2147483647	**`duration`**	integer	0 to 2147483647
infinity / unlimited	**`durationUnlimited`**	boolean	**`true`**

When supplying values:

- Only one of the two parameters is required
- The **`boolean`** is implicitly **`false`** if an **`int`** is supplied
- If the **`boolean`** is **`true`**, the **`int`** must not be supplied, or the Cisco TelePresence Server will return a fault

Media credits

Every participant that connects to a conference consumes a number of credits.

Note: The token requirements for a call cannot be known prior to instantiation of the call, so no checks are made on **`flex.participant.create`** or **`flex.participant.modify`** to determine if the call will have adequate resources. The client is therefore responsible for ensuring that the call has adequate resources.

The number of credits required for a given participant can be derived using the sum of the tokens (main video, extended video and audio) required for the participant and the **`mediaCreditTokenRanges`** array returned by flex.resource.query [p.91].

The array returned is effectively a conversion table from media credits to media tokens. For example, if the array returned is [48, 315, 630, 840, 1260, 2520, 3780, 5040, 7560, 10080], this can be interpreted as the following conversion table:

Table 5: Media credits to media tokens mapping

Media credits	Media tokens
48	0 to 48
315	49 to 315
630	49 to 630
840	631 to 840
1260	841 to 1260
2520	1261 to 2520
3780	2521 to 3780
5040	3781 to 5040
7560	5041 to 7560
10080	7561 to 10080

If media credit values supplied to API methods do not match any of the values in the "Media credits" column in the previous table, the value is rounded down to the next level. Supplying media credit values less than 48 allocates 0 credits.

Every participant must have enough credits to use the tokens configured for that participant. API methods fail if this requirement is not met. This applies to media credit values rounded down as described previously.

Participant calls are rejected if there are insufficient credits when connecting the call to the conference.

Media reservation

Reserved media resources are tokens and credits that have been assigned for exclusive use by a participant. Reservation guarantees that if an endpoint connection succeeds, media resources required to service the connection exist.

Note: The token requirements for a call cannot be known prior to instantiation of the call, so no checks are made on **`flex.participant.create`** or **`flex.participant.modify`** to determine if the call will have adequate resources. The client is therefore responsible for ensuring that the call has adequate resources.

Enumeration

This API supports incremental enumeration of objects such as conferences and participants. The following methods are typically associated with complete enumeration of a type of object:

- **`flex.`***object***`.enumerate`**
- **`flex.`***object***`.deletions.enumerate`**

Both methods use cookies to determine what content needs to be returned. To start the enumeration, the methods should be invoked without supplying a cookie. To continue the enumeration, the methods should be invoked with the cookie returned by the previous invocation.

Both methods return the boolean parameter **`moreAvailable`**. If the value of this parameter is **`true`**, more data is available.

For information on how you can use incremental enumeration to optimize resource usage, see flex.participant.media.enumerate [p.75].

The `.enumerate` methods

The **`.enumerate`** methods are intended for enumeration of live objects and return lists of *object* that are new or have been revised.

To use the **`.enumerate`** methods:

- On the first invocation, do not present a cookie. Information is returned on all live objects.
- On subsequent invocations, present a cookie. Information is returned on live objects that have changed or have been added since the previous invocation as indicated by the cookie.

The **`.enumerate`** methods may fail with **`Fault 102: 'cookie is invalid or expired'`** if the enumeration cannot be completed; that is, if all changes or additions that occurred since the last invocation cannot be listed. In such cases, restart both live and deletions enumerations, discarding the previous state.

The `.deletions.enumerate` methods

The **`.deletions.enumerate`** methods are intended for enumeration of objects that have been destroyed. They return lists of *object* IDs that have been deleted since the last invocation as indicated by the cookie.

To use the **`.deletions.enumerate`** methods:

- On the first invocation, do not present a cookie, No IDs are returned for the first invocation.
- On subsequent invocations, present a cookie. IDs of objects that have been deleted since the previous invocation of the method are returned.

The **`deletions.enumerate`** methods may fail with **`Fault 102: 'cookie is invalid or expired'`** if the enumeration cannot be completed; that is, if all deletions that occurred since the last invocation cannot be listed. In such cases, restart both live and deletions enumerations, discarding the previous state.

Enumeration method invocation

Typically, the enumeration methods should be invoked in response to feedback notification of an event. If the methods are invoked and no changes have occurred (since the last invocation as determined by the cookie), empty lists will be returned.

For example, to maintain a list of information about live conferences:

1. Invoke flex.conference.deletions.enumerate [p.53] with no parameters other than the Authentication [p.116] parameters, and store the **`cookie`** string parameter returned as the *deletions cookie*.
2. Invoke flex.conference.enumerate [p.54] with no parameters other than authentication parameters.
3. Go to step 5.
4. Invoke flex.conference.enumerate [p.54] with the **`cookie`** parameter set to the *live objects cookie*.
5. Store the **`cookie`** string parameter returned as the *live objects cookie*.
6. Process the list of conferences returned.

7. If **`moreAvailable`** is **`true`**, repeat from step 4.
8. Go to step 13.
9. Invoke flex.conference.deletions.enumerate [p.53] with the **`cookie`** parameter set to the *deletions cookie* (see above and also below).
10. Store the **`cookie`** parameter returned as the *deletions cookie*.
11. Process the **`conferenceIDs`** array.
12. If **`moreAvailable`** is **`true`**, repeat from step 9.
13. Wait for feedback notification.
14. In the event of a conference change or addition, go to step 4.
15. In the event of a conference deletion, go to step 9.

In the algorithm above, at the start of the enumeration, flex.conference.deletions.enumerate [p.53] is invoked before flex.conference.enumerate [p.54]. This ensures that the deletion of any conference returned by flex.conference.enumerate [p.54] will be returned by flex.conference.deletions.enumerate [p.53] when it occurs.

It is also possible that flex.conference.deletions.enumerate [p.53] will return the IDs of conferences which have not been returned by flex.conference.enumerate [p.54]. This can happen when a conference is created and destroyed before flex.conference.enumerate [p.54] is invoked or the enumeration has not proceeded far enough to return the conference ID.

Participants and participant media resources can be enumerated in a similar way using the following methods:

- flex.participant.enumerate [p.73] and/or flex.participant.media.enumerate [p.75]
- flex.participant.deletions.enumerate [p.72]

Feedback events and enumeration

Feedback events are generated to aid incremental enumeration of conferences and participants. See Feedback events [p.20] for more information.

For conferences:

- When a conference is created, modified or its state changes, the **`flexConferenceEnum`** event is generated. Invoke flex.conference.enumerate [p.54] to retrieve information for newly added or modified conferences. The **`flexConferenceEnum`** event is only generated for those modifications or state changes that affect data returned by flex.conference.enumerate [p.54].
- When a conference is destroyed, the **`flexConferenceDeletionsEnum`** event is generated. Invoke flex.conference.deletions.enumerate [p.53] to identify which conferences have been destroyed.

Similarly for participants:

- When a participant is created, modified or its state changes, the **`flexParticipantEnum`** event is generated. Invoke flex.participant.enumerate [p.73] to retrieve information for newly added or modified participants. The **`flexParticipantEnum`** event is only generated for those modifications or state changes that affect the data returned by the flex.participant.enumerate [p.73] method.
- When a participant is created or its media resource state changes, the **`flexParticipantMediaEnum`** event is generated. Invoke flex.participant.media.enumerate [p.75] to retrieve participant media information. The **`flexParticipantMediaEnum`** event is only generated for those modifications or state changes that affect the data returned by the flex.participant.media.enumerate [p.75] method.

- When a participant is destroyed, the **`flexParticipantDeletionsEnum`** event is generated. Invoke flex.participant.deletions.enumerate [p.72] to identify which participants have been destroyed.

DTMF

The set of valid characters for DTMF is:

- **`*#0123456789ABCD,`**

The comma character is used to insert delay. Each comma denotes a two-second delay.

Commands that take DTMF string parameters will accept any non-DTMF ASCII characters in the string but the TelePresence Server will ignore them; it processes the string until it reaches the end, sending only the tones for characters within the set **`*#0123456789ABCD`** and pausing the tone sequence by two seconds for each comma.

The TelePresence Server returns a fault if there are non-ASCII characters in the string.

Feedback receivers

The API allows you to register your application as a feedback receiver. This means that the application does not have to constantly poll the device if it wants to monitor activity. By using feedback events, you can avoid imposing the high loads that polling can cause especially when there are multiple API users.

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.

Note: The TelePresence Server expects your application to provide at least an **`HTTP 200 OK`** status header. The TelePresence Server logs a warning event if it cannot be sure your application received the feedback message.

- Use feedbackReceiver.configure [p.45] to register a receiver to listen for one or more Feedback events [p.20].
- Use feedbackReceiver.query [p.46] to return a list of receivers that are configured on the device.
- Use feedbackReceiver.reconfigure [p.47] to change the configuration of an existing feedback receiver.
- Use feedbackReceiver.remove [p.47] to remove an existing feedback receiver.
- Use feedbackReceiver.status [p.47] to display the status of a specific feedback receiver, and all the events to which it is subscribed.

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

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 **`feedbackReceiver.reconfigure`**. If it has not been set it 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:

Table 6: Feedback events

Event	Description
cdrAdded	One or more new Call Detail Records have been logged
configureAck	The source publishes this event to acknowledge that an application has successfully added, reconfigured, or removed a feedback receiver
deviceStatusChanged	Generated when the TelePresence Server is shut down or a feature key is added or removed. Invoke **`device.query`** for more details.
flexAlive	Alive notification. This event is generated every 10 seconds and should be used if it is a requirement to monitor whether the TelePresence Server is alive. See When to consider a TelePresence Server to be no longer alive.
flexConferenceDeletionsEnum	One or more conferences have been destroyed. Invoke flex.conference.deletions.enumerate [p.53] to get the identifiers of conferences destroyed.
flexConferenceEnum	The state of one or more conferences has changed. Invoke flex.conference.enumerate [p.54] to get the changes.

Table 6: Feedback events (continued)

Event	Description
flexResourceConfiguration	The resource configuration has changed. Media blades have been added or removed. Invoke flex.resource.query [p.91] to retrieve the new configuration.
flexParticipantAdvancedEnum	Participants have been created or their state has changed. Invoke flex.participant.advanced.enumerate [p.66] to get the changes.
flexParticipantDeletionsEnum	Participants have been destroyed. Invoke flex.participant.deletions.enumerate [p.72] to get the identifiers of participants destroyed.
flexParticipantEnum	Participants have been created or their state has changed. Invoke flex.participant.enumerate [p.73] to get the changes.
flexParticipantMediaEnum	Participants have been created or their media resources state has changed. Invoke flex.participant.media.enumerate [p.75] to get the changes.
flexResourceStatus	Resource usage has changed: calls, participants, conferences, media tokens, and media credits. Invoke flex.resource.status [p.93] to get the changes.
receiverDeleted	The feedback receiver receiving this event has been stopped and its configuration deleted or the URI of the feedback receiver has been changed, in which case this event is sent to the previous URI.
receiverModified	The feedback receiver receiving this event has been modified.
restart	The TelePresence Server has restarted or booted.

Note: When the URI of a feedback receiver is changed, **`receiverModified`** and **`receiverDeleted`** events are sent to the previous URI of the feedback receiver.

When to consider a TelePresence Server to be no longer alive

When monitoring the liveness of a TelePresence Server, it should be considered alive if the time since the last **`flexAlive`** feedback event does not exceed twice the feedback interval (that is, 20 seconds). After this time, the status of the server should be checked with flex.resource.status [p.93]; only if there is no response should the server be considered no longer alive.

Data structures and enumerated types

Enumerated types

Enumerated types as described here are a convenient way of describing the behavior of string fields for which arbitrary string values are not appropriate. Enumerated types are not an extension to the XML-RPC specification.

Each enumerated type has an associated list of strings. If a parameter's value is described as belonging to a particular enumerated type:

- Input strings that are not in the list will generate an invalid parameter fault.
- Only strings that are in the list will be returned by the TelePresence Server.
- The maximum length of the returned string is the same as the length of the longest string in the list.

Enumerated types described in this topic:

- Table 7: Access level enumerated type [p.22]
- Table 8: Motion vs Sharpness enumerated type [p.23]
- Table 9: Aspect ratio enumerated type [p.23]
- Table 10: Single screen layout enumerated type [p.23]
- Table 11: Multi-screen layout enumerated type [p.23]
- Table 12: Call protocol enumerated type [p.23]
- Table 13: Video format enumerated type [p.23]
- Table 14: Participant encryption enumerated type [p.24]
- Table 15: Video transmit size enumerated type [p.24]
- Table 16: Call conference state enumerated type [p.24]
- Table 17: Call state enumerated type [p.24]
- Table 18: Call protocol enumerated type [p.24]
- Table 19: Cascade roles enumerated type [p.25]
- Table 21: Switching mode enumerated type [p.25]
- Table 22: Participant connection state enumerated type [p.25]
- Table 23: Encryption status enumerated type [p.26]
- Table 24: Media status enumerated type [p.26]
- Table 25: Full screen mode enumerated type [p.26]
- Table 26: Audio gain mode enumerated type [p.27]
- Table 27: Control level enumerated type [p.27]

Table 7: Access level enumerated type

Name	Description
`chair`	The participant is granted chair access to the conference.
`guest`	The participant is granted guest access to the conference.

Table 8: Motion vs Sharpness enumerated type

Name	Description
`favorMotion`	Use high frame rates.
`favorSharpness`	Use high resolution.
`balanced`	Frame rate >= 12 fps.

Table 9: Aspect ratio enumerated type

Name	Description
`onlyFourToThree`	4:3 only.
`onlySixteenToNine`	16:9 only.
`allowAllResolutions`	Allow 4:3 as well as 16:9.

Table 10: Single screen layout enumerated type

Name	Description
`layoutSingle`	Full screen only.
`layoutActivePresence`	Active presence: full screen with pips.
`layoutProminent`	Single large pane and up to four small panes.
`layoutEqual`	Multiple panes of the same size.

Table 11: Multi-screen layout enumerated type

Name	Description
`layoutSingle`	Full screen only.
`layoutActivePresence`	Active presence: full screen with pips.

Table 12: Call protocol enumerated type

Name	Description
`h323`	H.323 Protocol.
`sip`	SIP (Session Initiation Protocol).

Table 13: Video format enumerated type

Name	Description
`NTSC`	National Television System Committee (NTSC) video format, fractions of 30 fps.
`PAL`	Phase Alternating Line (PAL) video format, fractions of 25 fps.

Table 14: Participant encryption enumerated type

Name	Description
forbidden	Encryption is denied.
required	Encryption is mandatory.
optional	Encryption is supported but not mandatory.

Table 15: Video transmit size enumerated type

Name	Description
none	Do not allow changes in video size during transmission.
dynamicResolution	Allow video size to be optimized during transmission.
dynamicCodecAndResolution	Allow video size to be optimized during transmission and/or dynamic codec selection.

Table 16: Call conference state enumerated type

Name	Description
idle	Initial state.
pinEntry	Call needs to enter PIN to progress.
blank	Showing blank video (and silent audio).
welcome	Showing conference welcome screen.
awaitingChair	Awaiting presence of one or more chair participants before activating.
awaitingAccept	Awaiting a chair participant to accept or deny the participation of the call in the conference.
complete	The call is now fully connected to the conference.

Table 17: Call state enumerated type

Name	Description
callStateIdle	Call is inactive.
callStateAlerting	Endpoint is ringing.
callStateAnswering	Call is in the process of being connected.
callStateConnected	Endpoint is connected.
callStateRetrying	Call is being retried.

Table 18: Call protocol enumerated type

Name	Description
h323	H.323 Protocol.
sip	SIP (Session Initiation Protocol).

Table 19: Cascade roles enumerated type

cascadeRole value	Description
cascadeNone	The participant is not a cascade link.
cascadeMaster	The participant is the master in a cascade link with at least one other TelePresence Server. This end is the central node in a star topology.
cascadeSlave	This participant is the slave in a cascade link to a master TelePresence Server. This end is the outer node in a star topology.

Table 20: Optimization profiles enumerated type

optimizationProfile value	Description
maximizeEfficiency	Screen licenses are conserved aggressively. This value gives the most calls for the available resources.
favorEfficiency	This is a balance of efficiency and experience that favors conserving screen licenses over attempting to grant the requested resolution.
favorExperience	Default. This is a balance of efficiency and experience that favors granting the requested resolution over conserving screen licenses.
maximizeExperience	Screen licenses are more readily allocated. This value gives the best experience of the four profiles. If you disable the optimization by bandwidth (by setting **optimizationProfile** to **capabilitySetOnly**), calls will be capable of higher resolutions at lower bandwidths but the inefficency in allocation could well outweigh the benefit.
capabilitySetOnly	This is the behavior of TelePresence Server 3.1. The TelePresence Server only considers the endpoint's maximum advertized resolution when reporting its screen license requirement to the managing system; it does not attempt to report resources based on the endpoint's advertized receive bandwidth.

Table 21: Switching mode enumerated type

displayLayoutSwitchingMode value	Description
switchingRoomSwitched	Room switching is used. When a participant from a room speaks, all segments (panels) from the room are shown on three-screen endpoints.
switchingSegmentSwitched	Segment switching is used. When a participant from a room speaks, only their segment(panel) of the room is shown on three-screen endpoints.

Table 22: Participant connection state enumerated type

connectionState value	Description
disconnected	All calls for this participant are disconnected
connecting	At least one of the participant's calls has not yet fully connected

Table 22: Participant connection state enumerated type (continued)

`connectionState` value	Description
`connected`	All the participant's calls are connected
`onHold`	The participant's calls are all connected, but the participant is on hold (not sending or receiving media)

Table 23: Encryption status enumerated type

`encryptionStatus` value	Description
`unknown`	The encryption status is not yet known because it is still being negotiated
`encrypted`	All of the associated channels are encrypted
`mixed`	Some of the associated channels are encrypted, others are unencrypted
`unencrypted`	All of the associated channels are unencrypted

Table 24: Media status enumerated type

`encryptionStatus` value	Description
`notNegotiated`	The associated channel has not been negotiated. If there are multiple channels, this status indicates that no channels in the associated category have been negotiated.
`inUse`	The associated channel is in use, or at least one channel in the category is in use.
`notInUse`	The associated channel has been negotiated but is not being used. If there are multiple channels, this status indicates that none of them are in use.
`muted`	The associated channel has been negotiated but is muted. If there are multiple channels, this status indicates that all of the channels in the associated category have been muted.

Table 25: Full screen mode enumerated type

Name	Description
`never`	The stream from the single-screen participant will never show in a full-screen pane when viewed on a multiscreen endpoint.
`always`	The stream from the single-screen participant is always allowed to show in a full-screen pane on a mutli-screen endpoint.
`dynamic`	The stream from the single-screen participant is allowed to show in a full-screen pane on a multiscreen endpoint, as for `allowed`. However, if there are other multiscreen endpoints to show, the single-screen participant will not show in a full-screen pane on a multiscreen endpoint. In this case, the view of the single-screen endpoint will be restricted to a smaller, continuous presence pane.

Table 26: Audio gain mode enumerated type

Name	Description
`gainModeDisabled`	No gain is applied to the audio.
`gainModeAutomatic`	The level of gain applied is calculated automatically to normalise audio levels in a conference.
`gainModeFixed`	The level of gain applied is a fixed value, supplied separately.

Table 27: Control level enumerated type

Name	Description
`controlNone`	The participant is not authorized to control any conference settings.
`controlLocal`	The participant is authorized to control local conference settings - i.e. those related to the participant's own endpoint - but is not authorized to control the conference settings that affect other participants.
`controlConference`	The participant is authorized to control conference settings that may affect other participants, such as locking the conference or disconnecting other participants, and is also authorized to control the local conference settings (such as changing the conference layout shown on the local endpoint).

Structs

The following structs are used by multiple methods in this API. Other structs that are applicable to one command alone are described with the command in the API command reference [p.36].

- Media tokens struct [p.27]
- Participant media resources struct [p.28]
- Table 30: callAttributes struct members [p.30]
- Conference URI details struct [p.34]
- Participant call definition struct [p.35]

Media tokens struct

Media tokens are used to describe how media resources should be used when assigned to conferences and participants.

Media token parameters are passed as a structure of the form described in the following tables. This struct is referred to as the **`mediaTokens`** struct in this document.

Table 28: Media tokens struct members

Parameter name	Type	Description
`total`	integer (>= 0)	**Required**. Maximum total media token usage permitted across all channels.

Table 28: Media tokens struct members (continued)

Parameter name	Type	Description
`maxPerChannel`	integer (>= 0, <= `total`)	Maximum media resource usage permitted for a channel. Default: see **`maxPerChannelUnlimited`**.
`maxPerChannelUnlimited`	boolean	Whether an unlimited number of resources can be used for a channel. See "Unlimited" integers [p.15]. Default: **`true`**.

When supplying this struct to the TelePresence Server:

- Negative values for **`maxPerChannel`** and **`total`** are invalid parameter values.
- If neither **`maxPerChannel`** nor **`maxPerChannelUnlimited`** is specified, **`maxPerChannelUnlimited`** is defaulted to **`true`**.
- If **`maxPerChannelUnlimited`** is **`true`**, the system media tokens per channel limit applies.
- The system media tokens per channel limit can be retrieved using the flex.resource.query [p.91] method and is the value of the **`maxMediaTokensPerChannel`** field.

When this struct is returned by the TelePresence Server:

- The field **`maxPerChannelUnlimited`** is only present if its value is set to **`true`**.
- The field **`maxPerChannel`** is only present when there is an upper limit on the number of resources that can be used and the value of **`maxPerChannelUnlimited`** is **`false`**.

Participant media resources struct

The collection of parameters in the participant media resources struct describe the media resource configuration for a participant.

This struct is referred to as the **`participantMediaResources`** struct in this document.

Table 29: Participant media resources struct members

Parameter name	Type	Description
`mediaTokensMainVideo`	struct	Media token values representing the maximum resources that can be assigned to main video within a participant. See Media tokens struct [p.27].
`mediaTokensExtendedVideo`	struct	Media token values representing the maximum resources that can be assigned to extended video within a participant. See Media tokens struct [p.27].
`mediaTokensAudio`	struct	Media token values representing the maximum resources that can be assigned to audio within a participant. See Media tokens struct [p.27].
`numMediaCredits`	integer (>= 0)	Number of credits configured for the participant. See Media credits [p.15].

All members of the **`participantMediaResources`** struct are mandatory. In practice, this means that the following parameters must always be present:

- **`participantMediaResources.mediaTokensMainVideo.total`**
- **`participantMediaResources.mediaTokensExtendedVideo.total`**
- **`participantMediaResources.mediaTokensAudio.total`**
- **`participantMediaResources.numMediaCredits`**

When **`participantMediaResources`** fields are updated, all members of the struct are changed. Unspecified optional fields are set to their default values. For example, if media resource usage per channel is not specified in the update, default values are applied and it will be set to unlimited.

Note: The token requirements for a call cannot be known prior to instantiation of the call, so no checks are made on **`flex.participant.create`** or **`flex.participant.modify`** to determine if the call will have adequate resources. The client is therefore responsible for ensuring that the call has adequate resources.

`participantMediaResources.numMediaCredits` must be greater than or equal to the sum of media token values and is subject to the restrictions described in Media credits [p.15].

`participantMediaResources` can be configured at multiple points in the API. The media resources for a participant are selected in the following order of preference:

1. Participant specification, if defined (participant creation and participant modification).
2. Conference URIs, if defined.
3. Conference default specification (always defined).

Call attributes struct

Where call attributes are accepted:

Call attributes can be specified in the following places, by supplying a **`callAttributes`** struct as described in Table 30: callAttributes struct members [p.30]:

- Participant specification (see flex.participant.create [p.70] command).
- Participant modification (see flex.participant.modify [p.77] command).
- Conference URI specification (seeConference URI details struct [p.34]).
- Conference creation (see flex.conference.create [p.49] command).
- Conference modification (see flex.conference.modify [p.57] command).

How call attributes are derived

On conference creation you may supply a **`callAttributes`** struct, whose values then become the default for any conference URIs or participants that you subsequently create. If you don't supply the struct at that point, or omit any of its optional members, then the omitted parameters will take on the default values (as listed in Table 30: callAttributes struct members [p.30]) whenever they are subsequently used or returned by the TelePresence Server.

You may also supply **`callAttributes`** when you create or modify conference URIs or participants. Whenever there is overlap between call attributes, the attributes specified 'nearest' the participant will take precedence, as follows:

- Call attributes supplied for a participant will take precedence over those supplied for a conference URI or a conference

- Call attributes specified for a conference URI will take precedence over those specified for the conference
- Wherever you supply an attribute it will take precedence over the default

The values of all **`callAttributes`** fields are set when a participant is instantiated. This means that subsequent changes to the conference's call attributes or conference URI's call attributes have no effect on existing participants.

The following table lists the parameters that are accepted by this struct.

Table 30: **`callAttributes`** struct members

Parameter name	**Type**	**Description**
`accessLevel`	string	Access level associated with this participant, when connected. See Table 7: Access level enumerated type [p.22]. Default: **`chair`**.
`encryption`	string	Encryption setting. See Table 14: Participant encryption enumerated type [p.24]. Default: **`optional`**.
`autoDisconnect`	boolean	Whether this call automatically disconnects if the only calls connected to a conference have **`autoDisconnect`** set. Default: **`false`**.
`maxTransmitPacketSize`	integer (428–1448)	Sets the maximum size (bytes) of video packets sent by the TelePresence Server, including IP headers. **Note:** In TelePresence Server 3.1 and earlier versions, this setting did not include IP headers. When upgrading to 4.0, the TelePresence Server retains the previous setting which effectively reduces the maximum video data size by the size of the IP headers. If you want to set the maximum possible size for the video packets, use 1428 for an IPv4 network or 1448 for an IPv6 network. We recommend using the default (1400), or higher, unless there is a known packet size restriction in the path. This allows the TelePresence Server to make the most efficient use of the available bandwidth. If the packets are too large for a network that requires a smaller maximum transmission unit (MTU), network elements may fragment and reintegrate the packets which can impair performance.
`packetLossThreshold`	integer (0–100)	Packet losses are reported when the proportion of packets lost (percentage * 10) exceeds this threshold over a short period of time. Default: 0.
`videoRxFlowControlOnErrors`	boolean	Flow control on video errors. Default: **`true`**.

Table 30: callAttributes struct members (continued)

Parameter name	Type	Description
videoRxFlowControlOnViewedSize	boolean	Flow control based on viewed size. Default: **true**. This parameter is modified in version 4.0 as follows: if **true**, flow control is applied only when the video stream is not viewed by anyone. Formerly, this optimization would also apply when the stream was viewed at small resolutions.
videoTxSizeOptimization	string	Video transmit size setting. See Table 15: Video transmit size enumerated type [p.24]. Default: **dynamicCodecAndResolution**.
presentationContributionAllowed	boolean	Whether the endpoint can contribute presentations to a conference. Default: **true**.
presentationTakeoverAllowed	boolean	Whether the endpoint can start contributing presentations even if the conference already has an active presentation. Default: **true**.
videoTxPresentationAllowed	boolean	Whether the endpoint can receive presentations in its extended video channel (if available). Default: **true**.
videoTxPresentationMainVideoAllowed	boolean	Whether the endpoint can receive presentations in its main video channel (if an extended video channel is unavailable, disabled, or there are insufficient resources available). For multi-call participants, this option is always **false**. Setting it to **true** will result in it implicitly being set to **false**. Any implicit changes will be reflected in the return values of flex.participant.query [p.77]. Default: **true**.
audioStereoEnabled	boolean	Whether support for stereo is enabled. Default: **true**.
audioDirectionalEnabled	boolean	Whether directional audio is enabled. Default: **true**.
indicateUnencryptedParticipants	boolean	Whether the unencrypted icon is displayed. Default: **true**.
indicateAudioOnlyParticipants	boolean	Whether the audio only icon is displayed. Default: **true**.
mainVideoTxPictureAspectRatio	string	Permissible aspect ratios for the main video channel. See Table 9: Aspect ratio enumerated type [p.23]. Default: **onlySixteenToNine**.
extendedVideoTxPictureAspectRatio	string	Permissible aspect ratios for the extended video channel. See Table 9: Aspect ratio enumerated type [p.23]. Default: **allowAllResolutions**.
videoTxFormat	string	Video format. See Table 13: Video format enumerated type [p.23]. Default: **NTSC**.
videoTxMotionSharpness	string	Motion sharpness setting. See Table 8: Motion vs Sharpness enumerated type [p.23]. Default: **balanced**.

Table 30: callAttributes struct members (continued)

Parameter name	Type	Description
videoRxClearVisionEnabled	boolean	Whether upscaling using ClearVision is allowed. Default: **true**.
video60fpsEnabled	boolean	Whether support for 60 frames per second is enabled. Default: **true**.
fullScreenMode	string	Setting for full screen viewing of single-screen endpoints. See Table 25: Full screen mode enumerated type [p.26]. Default: **always**.
displaySelfView	boolean	Whether participants see themselves as well as other participants. Default: **false**.
displayShowBorders	boolean	Whether panes are surrounded with a border to separate them visually. Default: **true**.
displayDefaultLayoutSingleScreen	string	Layout scheme used for single-screen endpoints. See Table 10: Single screen layout enumerated type [p.23]. Default: **layoutActivePresence**.
displayDefaultLayoutMultiScreen	string	Layout scheme used for multiscreen endpoints. See Table 11: Multi-screen layout enumerated type [p.23]. Default: **layoutActivePresence**.
displayForceDefaultLayout	boolean	If **true**, this prevents the participant from changing the display layout using FECC, DTMF, or ActiveControl. The layout sent to the endpoint will be forced to the configured value for that type of endpoint; that is, the value of either **displayDefaultLayoutSingleScreen** or **displayDefaultLayoutMultiScreen**. Default: **false**.
displayShowEndpointNames	boolean	Whether endpoint names are shown as panel labels. Default: **false**.
displayHighlightActiveSpeaker	boolean	Whether the active speaker is highlighted with a distinctive border. Default: **true**.
audioReceiveGainMode	string	See Table 26: Audio gain mode enumerated type [p.27]. Either **gainModeDisabled**, **gainModeAutomatic**, or **gainModeFixed**. Default is **gainModeAutomatic**.
audioReceiveGain	integer (-12000 to +12000)	Gain for received audio when **audioReceiveGainMode** is **gainModeFixed**. The unit is millidecibels and the default value is **0**. This call attribute is not required for other values of **audioReceiveGainMode** and will be ignored if supplied in these cases.
audioTransmitGain	integer (-12000 to +12000)	Gain for transmitted audio, measured in millidecibels. Default: 0.

Table 30: callAttributes struct members (continued)

Parameter name	Type	Description
`forceTIP`	boolean	Defines whether the use of TIP (Telepresence Interoperability Protocol) is enforced, even if the endpoint does not indicate that it is TIP capable. Default: **`false`**. This legacy setting is kept in the API to support legacy TIP endpoints. We do not recommend using it in any other circumstances because this could result in reduced functionality.
`audioRxStartMuted`	boolean	Whether audio from the endpoint is muted at the start of a call. Default: **`false`**.
`videoRxStartMuted`	boolean	Whether video from the endpoint is muted at the start of a call. Default: **`false`**.
`audioTxStartMuted`	boolean	Whether audio to the endpoint is muted at the start of a call. Default: **`false`**.
`videoTxStartMuted`	boolean	Whether video to the endpoint is muted at the start of a call. Default: **`false`**.
`autoReconnect`	boolean	Whether outgoing calls dropped for abnormal reasons are reconnected. Not effective for incoming calls. This setting will be ignored if **`alwaysReconnect`** is **`true`**. Default: **`false`**.
`recordingDevice`	boolean	Whether this call is treated as a recording device by muting received video and displaying a recording icon on other endpoints. Default: **`false`**.
`recordingDeviceIndicateOnly`	boolean	Whether this call is indicated as a recording device by displaying a recording icon on other endpoints. Received video is not muted. Default: **`false`**.
`deferConnect`	boolean	Applies only to calls dialed out from the TelePresence Server. If **`true`**, the TelePresence Server defers connecting this call until at least one other call is connected to the conference. If **`false`**, the TelePresence Server connects the call as soon as the conference starts. Default: **`false`**.
`alwaysReconnect`	boolean	If **`true`**, the TelePresence Server will reconnect this call in all disconnection circumstances. Applies only to calls dialed out from the TelePresence Server. Default: **`false`**. **CAUTION:** This feature is intended for reconnecting integrated systems. Do not use it directly with user endpoints, as they will always be redialed even after deliberate disconnection.
`iXEnabled`	boolean	Defines whether the iX protocol is enabled on this call. This attribute is effective from the start of the call and cannot be changed during the call. Default: **`false`**.

Table 30: callAttributes struct members (continued)

Parameter name	Type	Description
displayLayoutSwitchingMode	string	Defines how the display of this multicamera system is switched into view on three-screen endpoints. Either **switchingRoomSwitched** or **switchingSegmentSwitched**. Defaults to **switchingSegmentSwitched**. See Table 21: Switching mode enumerated type [p.25].
indicateMuting	boolean	Defines whether the participant is shown an icon representing that their audio has been muted on the TelePresence Server. **true** shows the icon when the participant has been muted. Default: **true**.
allowStarSixMuting	boolean	Defines whether participants can mute or unmute their audio by pressing *6. **true** allows the participant to use the *6 combination to mute/unmute. Default: **true**.

Conference URI details struct

The conference URI details struct defines a conference URI and its associated access levels and media resources.

Table 31: Conference URI details struct members

Parameter name	Type	Description
URI	string (80)	**Required**. String used by endpoints to connect to this conference. See Conference URI identifiers [p.12].
callBandwidth	integer (>= **minCallBandwidth** and <= **maxCallBandwidth**)	**Required**. Connection bandwidth measured in bits per second. See flex.resource.query [p.91].
PIN	string (40)	PIN for the conference at this URI and access level. Participants will only need to supply this PIN when calling in to the associated conference URI. A PIN is never requested when the TelePresence Server calls out to an endpoint. Default: empty.
callAttributes	struct	This Call attributes struct [p.29] contains attributes applied to calls connecting to a conference using the aforementioned URI. See Call attributes struct [p.29] and How call attributes are derived [p.29]. Default: inherits the conference default call attributes, see flex.conference.create [p.49] and flex.conference.modify [p.57].
participantMediaResources	struct	The Participant media resources struct [p.28] contains parameters that define the participant media resource configuration.

If **`participantMediaResources`** settings are absent, settings from the conference default **`participantMediaResources`** apply.

Participant call definition struct

A single participant call can be defined as one of incoming or outgoing, but not both. As a result, there are two call definition structs: one for incoming calls and the other for outgoing calls.

Table 32: Incoming participant call definition struct members

Parameter name	Type	Description
`URI`	string (80)	**Required**. String used to connect to this conference. See Participant conference URIs [p.13].
`callBandwidth`	integer (>= **`minCallBandwidth`** and <= **`maxCallBandwidth`**)	**Required**. Connection bandwidth measured in bits per second. See flex.resource.query [p.91].
`disconnectOnIncoming`	boolean	Whether the existing call (if one exists) is disconnected when an incoming call comes through the **`URI`**. Default: **`false`**.

Table 33: Outgoing participant call definition struct members

Parameter name	Type	Description
`remoteAddress`	string (80)	**Required**. Address of the endpoint expressed in the form of an endpoint address or E164 number.
`protocol`	string	**Required**. Call control protocol for outgoing call only. See Table 18: Call protocol enumerated type [p.24].
`callBandwidth`	integer (>= **`minCallBandwidth`** and <= **`maxCallBandwidth`**)	**Required**. Connection bandwidth in bits per second. See flex.resource.query [p.91].

API command reference

This section contains a reference to each of the commands available when the operation mode is set to **`flexible`**.

The commands are grouped alphabetically by the objects that they query or modify. The following information is provided for each command:

- Description of the command's effect
- Accepted parameters, and whether they are required
- Returned parameters, and whether they are conditionally returned

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

- callHome.configure [p.37]
- callHome.query [p.37]
- cdrlog.enumerate [p.38]
- cdrlog.query [p.39]
- device.feature.add [p.40]
- device.feature.remove [p.40]
- device.health.query [p.40]
- device.network.query [p.41]
- device.query [p.43]
- device.restart [p.44]
- device.restartlog.query [p.44]
- feedbackReceiver.configure [p.45]
- feedbackReceiver.query [p.46]
- feedbackReceiver.reconfigure [p.47]
- feedbackReceiver.remove [p.47]
- feedbackReceiver.status [p.47]
- flex.call.status [p.48]
- flex.conference.create [p.49]
- flex.conference.deletions.enumerate [p.53]
- flex.conference.destroy [p.54]
- flex.conference.enumerate [p.54]
- flex.conference.getMetadata [p.56]
- flex.conference.modify [p.57]
- flex.conference.query [p.61]
- flex.conference.sendUserMessage [p.64]
- flex.conference.sendWarning [p.65]
- flex.conference.status [p.65]
- flex.participant.advanced.enumerate [p.66]
- flex.participant.call.disconnect [p.69]
- flex.participant.clearImportant [p.70]

- flex.participant.create [p.70]
- flex.participant.deletions.enumerate [p.72]
- flex.participant.destroy [p.73]
- flex.participant.enumerate [p.73]
- flex.participant.media.enumerate [p.75]
- flex.participant.modify [p.77]
- flex.participant.query [p.77]
- flex.participant.requestDiagnostics [p.79]
- flex.participant.requestPreview [p.85]
- flex.participant.sendDTMF [p.87]
- flex.participant.sendUserMessage [p.88]
- flex.participant.setImportant [p.88]
- flex.participant.setMute [p.88]
- flex.participant.status [p.89]
- flex.resource.query [p.91]
- flex.resource.status [p.93]
- system.info [p.93]

callHome.configure

Configures the TelePresence Server to automatically report diagnostic data to Cisco's Call Home service. This feature is disabled by default, but we strongly recommend that you enable it to ensure the best possible support for your device.

Note: The TelePresence Server currently only supports anonymous reporting.

Table 34: `callHome.configure` inputs

Parameter name	Type	Description
`mode`	string	Set the Call Home mode. One of `disabled` or `anonymous`. Can only be set to `anonymous` if the encryption feature key is present. Defaults to `disabled` if it has never been configured. Omit the parameter to leave the current setting unchanged.
`automatic`	boolean	Controls automatic Call Home. `true` enables automatic Call Home. `false` disables automatic Call Home. Only has effect when mode is `anonymous`. Omit the parameter to leave the current setting unchanged.

callHome.query

Queries the TelePresence Server to retrieve its Call Home configuration. This feature reports diagnostic data to Cisco's Call Home service.

Note: The TelePresence Server currently only supports anonymous reporting.

Table 35: **`callHome.query`** returned data

Parameter name	Type	Description
`mode`	string	Call Home mode. One of **`disabled`** or **`anonymous`**. Defaults to **`disabled`** if it has never been configured.
`automatic`	boolean	**`true`** if automatic Call Home is enabled. **`false`** if automatic Call Home is disabled. Only has effect if mode is **`anonymous`**. Defaults to **`false`** if it has never been configured.

cdrlog.enumerate

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.

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.

Table 36: **`cdrlog.enumerate`** optional or conditional inputs

Parameter name	Type	Description
`index`	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, the device will enumerate events from the beginning of the CDR log.
`numEvents`	integer	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. Fewer events are returned if they are too large to fit into a single response. Clients should look at the **`eventsRemaining`** parameter in the response and re-enumerate, starting from **`nextIndex`**, if necessary.
`filter`	array	An array of strings, which contain the names of event types by which to filter the response. Omit **`filter`** to return all event types or include a subset of the following: **`conferenceStarted`**, **`conferenceFinished`**, **`conferenceActive`**, **`conferenceInactive`**, **`participantConnected`**, **`participantJoined`**, **`participantMediaSummary`**, **`participantLeft`**, **`participantDisconnected`**.

Table 37: **`cdrlog.enumerate`** returned data

Parameter name	Type	Description
`startIndex`	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.

Table 37: cdrlog.enumerate returned data (continued)

Parameter name	Type	Description
nextIndex	integer	Revision number of the data being provided, reusable in a subsequent call to the API.
eventsRemaining	boolean	If **true**, there is more data in the requested enumeration than has been returned in this response.
currentTime	dateTime.iso8601	The system's current time (UTC).
events	array of structs	Each member of the array is a struct that represents a recorded event. The structures all have some common fields (**time**, **type**, **index**) and may have other fields that are specific to the event type.

Events array

The following parameters are common to all CDR log events, but each struct will also contain parameters specific to the event type. See *Cisco TelePresence Conferencing Call Detail Records File Format Reference Guide* for details of all the TelePresence Server's event types.

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

Table 38: Common CDR log event parameters

Parameter name	Type	Description
time	dateTime.iso8601	Date and time when the event was logged; for example, **20110119T13:52:42**.
type	string	Name of the event type.
index	integer	Index of the CDR log message.

Note: The *Cisco TelePresence Conferencing Call Detail Records File Format 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.

cdrlog.query

Returns information about the CDR log. This command takes no input parameters.

Table 39: **cdrlog.query** returned data

Parameter name	Type	Description
firstIndex	integer	Index of the oldest stored event.
numEvents	integer	Total number of events stored.

device.feature.add

Adds a license or feature to the TelePresence Server. You need to obtain a key from Cisco or one of its resellers prior to running this command.

Table 40: **`device.feature.add`** inputs

Parameter name	Type	Description
`key`	string	**Required**. Use this unique code when you wish to add conferencing capacity or an optional feature to your TelePresence Server.

device.feature.remove

Removes a license or feature from the TelePresence Server. Use **`device.query`** to read the keys from a TelePresence Server.

Table 41: **`device.feature.remove`** inputs

Parameter name	Type	Description
`key`	string	**Required**. The unique code associated with the optional feature or license that you wish to remove from the TelePresence Server.

device.health.query

Returns the current status of the device, such as health monitors and CPU load. This command takes no input parameters.

Table 42: **`device.health.query`** returned data

Parameter name	Type	Description
`cpuLoad`	integer	CPU load expressed as a percentage of the maximum.
`fanStatus`	string	**`ok`** or **`outOfSpec`**. This parameter is returned only on appliances, eg. Media 310 or TelePresence Server 7010, which have their own fans. This parameter is not returned for TelePresence Server blades.
`fanStatusWorst`	string	Worst fan status recorded on this device since it restarted. One of **`ok`** or **`outOfSpec`**. This parameter is returned only on appliances, eg. Media 310 or TelePresence Server 7010, which have their own fans. This parameter is not returned for TelePresence Server blades.
`temperatureStatus`	string	One of **`ok`** (the temperature is currently within the normal operating range), **`outOfSpec`** (the temperature is currently outside the normal operating range), or **`critical`** (the temperature is too high and the device will shutdown if this condition persists).

Table 42: device.health.query returned data (continued)

Parameter name	Type	Description
temperatureStatusWorst	string	Worst temperature status recorded on this device since it booted. One of **ok**, **outOfSpec**, or **critical**.
rtcBatteryStatus	string	Current status of the RTC battery (Real Time Clock). One of **ok**, **outOfSpec**, or **critical**.
rtcBatteryStatusWorst	string	Worst status of the RTC battery (Real Time Clock) recorded on this device since it booted. One of **ok**, **outOfSpec**, or **critical**.
voltagesStatus	string	One of **ok** (the voltage is currently within the normal range), **outOfSpec** (the voltage is currently outside the normal range), or **critical**.
voltagesStatusWorst	string	Worst voltage status recorded on this device since it booted. One of **ok**, **outOfSpec**, or **critical**.
operationalStatus	string	One of **active** (the device is active), **shuttingDown** (the device is shutting down), **shutDown** (the device has shut down), or **unknown**.

device.network.query

Queries the device for its network information. The call takes no parameters and returns the following data structures. Some of the data listed below will be omitted if the interface is not enabled or configured. The query returns empty strings or dashes for addresses that are not configured.

Note: Packet counts and other statistics are measured with 32-bit signed integers, and may therefore wrap.

Table 43: **device.network.query** returned data

Parameter name	Type	Description
portA	struct	The struct contains configuration and status information for Ethernet port A on the device. See Table 44: Port struct members [p.41].
dns	array of structs	Each member of the array is a struct representing a set of DNS parameters for the queried device. See Table 45: DNS struct members [p.43].

Table 44: Port struct members

Parameter name	Type	Description
enabled	boolean	Whether the port is enabled.
ipv4Enabled	boolean	Whether IPv4 interface is enabled. Always returned unless there are no IP interfaces enabled on the port (neither IPv4 nor IPv6 is enabled).
ipv6Enabled	boolean	Whether IPv6 interface is enabled. Always returned unless there are no IP interfaces enabled on the port (neither IPv4 nor IPv6 is enabled).
linkStatus	boolean	Whether the Ethernet connection to this port is active.

Table 44: Port struct members (continued)

Parameter name	Type	Description
speed	integer	Speed of the connection on this Ethernet port. One of 10, 100 or 1000, in Mbps.
fullDuplex	boolean	Whether the port can support a full-duplex connection.
macAddress	string	MAC address of this port. A 12-character string of hex digits with no separators.
packetsSent	integer	Number of packets sent from this Ethernet port.
packetsReceived	integer	Number of packets received on this Ethernet port.
multicastPacketsSent	integer	Number of multicast packets sent from this Ethernet port.
multicastPacketsReceived	integer	Number of multicast packets received on this Ethernet port.
bytesSent	integer	Number of bytes sent by the device.
bytesReceived	integer	Number of bytes received by the device.
queueDrops	integer	Number of packets dropped from the queue on this network port.
collisions	integer	Count of the network collisions recorded by the device.
transmitErrors	integer	Count of transmission errors on this Ethernet port.
receiveErrors	integer	Count of receive errors on this port.
bytesSent64	string	64-bit versions of the **bytesSent** statistic expressed as a string rather than an integer.
bytesReceived64	string	64-bit versions of the **bytesReceived** statistic expressed as a string rather than an integer.
dhcpv4	boolean	Whether the ipv4 address is allocated by DHCP. Not returned if not configured.
ipv4Address	string	IPv4 address in the dotted quad format. Not returned if not configured.
ipv4SubnetMask	string	IPv4 subnet mask in the dotted quad format. Not returned if not configured.
defaultipv4Gateway	string	IPv4 address in the dotted quad format. Not returned if not configured.
ipv6Address	string	IPv6 address in CIDRformat. Not returned if not configured.
ipv6Conf	string	Indicates how the IPv6 address is assigned. One of **automatic** (IPv6 address is configured by SLAAC/DHCPv6) or **manual** (IPv6 address is configured manually). Not returned if not configured.
ipv6PrefixLength	integer	Length of the IPv6 address prefix. Not returned if not configured.
defaultIpv6Gateway	string	Address of the IPv6 default gateway in CIDR format. Not returned if not configured.

Table 44: Port struct members (continued)

Parameter name	Type	Description
linkLocalIpv6Address	string	Link local IPv6 address in CIDR format. Not returned if not configured.
linkLocalIpv6PrefixLength	integer	Length of the link local IPv6 address prefix. Not returned if not configured.

Table 45: DNS struct members

Parameter name	Type	Description
hostName	string	Host name of the queried device.
nameServer	string	IP address of the name server, in dotted quad format (IPv4) or CIDR format (IPv6).
nameServerSecondary	string	IP address of the secondary name server, in dotted quad format (IPv4) or CIDR format (IPv6).
domainName	string	Domain name of the queried device (DNS suffix).

device.query

Returns high level status information about the device. This command takes no input parameters.

Table 46: **device.query** returned data

Parameter name	Type	Description
currentTime	dateTime.iso8601	The system's current date and time.
restartTime	dateTime.iso8601	The system's date and time when it started.
uptime	integer	The difference, in seconds, between the system's current time and the system's restart time.
serial	string	Serial number of this device.
apiVersion	string	Version number of the API implemented by this TelePresence Server.
activatedFeatures	array of structs	Each member of the array is a struct, representing an active feature. See Table 47: Active feature struct members [p.44].
activatedLicenses	array of structs	Each member of the array is a struct, representing an active license. See Table 48: Active license struct members [p.44].
shutdownStatus	string	Displays one of the following: **notShutdown**, **shutdownInProgress**, **shutdown**, or **error**.
mediaResourceRestarts	integer	The count of unexpected restarts that have occurred on the device's media resources (signal processor chips).

Table 47: Active feature struct members

Parameter name	Type	Description
`feature`	string	The name of the feature, eg. **`Encryption`**.
`key`	string	The unique code associated with the feature.
`expiry`	dateTime.iso8601	The time at which this temporary key will expire. **`expiry`** is not present for permanent keys.

Table 48: Active license struct members

Parameter name	Type	Description
`license`	string	The name of the license.
`ports`	integer	The number of screen licenses provided by this license.
`key`	string	The unique code associated with the license.
`expiry`	dateTime.iso8601	The time at which this temporary key will expire. **`expiry`** is not present for permanent keys.

device.restart

Restarts the device, or shuts it down without a restart. This command does not return any parameters.

Table 49: **`device.restart`** input parameters

Parameter name	Type	Description
`shutdownOnly`	boolean	(Optional) Set to **`true`** to shut down without restarting. Default: **`false`**.

device.restartlog.query

Returns the restart log - also known as the system log on the web interface. This command takes no input parameters.

Table 50: **`device.restartlog.query`** returned data

Parameter name	Type	Description
`log`	array of structs	Each member of the array is a struct containing a restart **`reason`**. See Table 51: Log struct members [p.44]. This information source is called "system log" in the web interface.

Table 51: Log struct members

Parameter name	Type	Description
`time`	dateTime.iso8601	Date and time of the restart.

Table 51: Log struct members (continued)

Parameter name	Type	Description
`reason`	string	Reason for the device restart. See Table 52: Restart reason enumerated type [p.45].

Table 52: Restart **`reason`** enumerated type

`reason` value	Description
User requested shutdown	The device restarted normally after a user initiated a shutdown.
User requested reboot from web interface	The device restarted itself because a user initiated a reboot via the web interface.
User requested upgrade	The device restarted itself because a user initiated an upgrade.
User requested reboot from console	The device restarted itself because a user initiated a reboot via the console.
User requested reboot from API	The device restarted itself because a user initiated a reboot via the API.
User requested reboot from FTP	The device restarted itself because a user initiated a reboot via FTP.
User requested shutdown from supervisor	The device restarted normally after a user initiated a shutdown from the supervisor.
User requested reboot from supervisor	The device restarted itself because a user initiated a reboot via the supervisor.
User reset configuration	The device restarted itself because a user reset the configuration.
Cold boot	The device restarted itself because a user initiated a cold boot.
unknown	The software is unaware why the device restarted.

feedbackReceiver.configure

Configures the device to send feedback about the specified **`subscribedEvents`** to the specified **`receiverURI`**.

Table 53: **`feedbackReceiver.configure`** inputs

Parameter name	Type	Description
`receiverURI`	string (255)	**Required**. Fully-qualified **`http`** or **`https`** URI (for example, **`http://tms1:8080/RPC2`**) to which feedback events are sent. If no port number is specified, the device uses the protocol defaults (80 and 443 respectively).

Table 53: feedbackReceiver.configure inputs (continued)

Parameter name	Type	Description
receiverIndex	integer (< 0, or 1–20 inclusive)	Index of the feedback receiver indicating the slot that this receiver should use. A negative value indicates that the feedback receiver should use any available slot (preferred). Default: 1. **Note:** The default **receiverIndex** is 1, and will always overwrite a feedback receiver in the first index position. You should query the device first, or use a negative value, if you want to be certain not to overwrite an existing feedback receiver.
sourceIdentifier	string (255) ASCII characters only	Identifier string for the receiver. The originating device uses this parameter to identify itself to the listening receiver (or receivers). If the parameter is not explicitly set, the device identifies itself with the MAC address of its Ethernet port A interface. Default: empty.
subscribedEvents	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. See Feedback events [p.20]. If this array is absent, the receiver subscribes to all notifications by default. Default: all events.

Table 54: **feedbackReceiver.configure** returned data

Parameter name	Type	Description
receiverIndex	integer	Position of this feedback receiver in the device's table of feedback receivers.

feedbackReceiver.query

Requests a list of all the feedback receivers that have previously been configured for the device. It does not accept parameters other than the authentication strings. If there are no feedback receivers to enumerate, **feedbackReceiver.query** returns an empty **receivers** array.

Table 55: **feedbackReceiver.query** returned data

Parameter name	Type	Description
receivers	array	Array of feedback receivers, with members corresponding to the entries in the receivers table on the web interface of the device.

Table 56: Feedback receiver struct members

Parameter name	Type	Description
index	integer (1–20)	Position of this feedback receiver in the table of feedback receivers. The index number is also the feedback receiver ID.

Table 56: Feedback receiver struct members (continued)

Parameter name	Type	Description
sourceIdentifier	string (255) ASCII characters only	Source identifier string, which can be empty. The originating device uses this parameter to identify itself to the listening receiver (or receivers). If the parameter is not explicitly set, the device identifies itself with the MAC address of its Ethernet port A interface.
receiverURI	string (255)	Fully-qualified **http** or **https** URI (for example, **http://tms1:8080/RPC2**) to which feedback events are sent.

feedbackReceiver.reconfigure

Overwrites the configuration of an existing feedback receiver with any parameters that you supply. The TelePresence Server keeps the current configuration for any parameters that you do not specify.

Table 57: **feedbackReceiver.reconfigure** inputs

Parameter name	Type	Description
receiverIndex	integer (1–20)	**Required**. Index of the feedback receiver to be reconfigured. The call returns a fault if there is no feedback receiver at the specified **receiverIndex**.
receiverURI	string (255)	Fully-qualified **http** or **https** URI (for example, **http://tms1:8080/RPC2**) to which feedback events are sent. If omitted, the device uses the originally configured **receiverURI**.
sourceIdentifier	string (255) ASCII characters only	Identifier string for the receiver. The originating device uses this parameter to identify itself to the listening receiver (or receivers). If omitted, the device uses the originally configured **sourceIdentifier**.
subscribedEvents	array	Array of strings identifying the events to which the receiver subscribes. See Feedback events [p.20]. If omitted, the event notifications set in the original configuration request remain unchanged.

feedbackReceiver.remove

Removes the specified feedback receiver. This command returns no data.

Table 58: **feedbackReceiver.remove** inputs

Parameter name	Type	Description
receiverIndex	integer (1–20)	**Required**. Index of the feedback receiver to be removed.

feedbackReceiver.status

Asks the device for a list of all the events to which a feedback receiver subscribes.

Table 59: **`feedbackReceiver.status`** inputs

Parameter name	Type	Description
`receiverIndex`	integer (1–20)	**Required**. Index of the feedback receiver.

Table 60: **`feedbackReceiver.status`** returned data

Parameter name	Type	Description
`receiverIndex`	integer (1–20)	Index of the feedback receiver entry, which also serves as the feedback receiver ID.
`sourceIdentifier`	string (255) ASCII characters only	Identifier string for the receiver. The originating device uses this parameter to identify itself to the listening receiver (or receivers). If the parameter is not explicitly set, the device identifies itself with the MAC address of its Ethernet port A interface.
`receiverURI`	string (255)	Fully-qualified **`http`** or **`https`** URI (for example, **`http://tms1:8080/RPC2`**) to which feedback events are sent.
`subscribedEvents`	array	Array of strings identifying the event names that are enabled for this feedback receiver. See Feedback events [p.20].

flex.call.status

Returns the status of the specified call.

Table 61: **`flex.call.status`** inputs

Parameter name	Type	Description
`callID`	string (50)	**Required**. Call identifier assigned by the TelePresence Server. See Identifiers and client references [p.11].

Table 62: **`flex.call.status`** returned data

Parameter name	Type	Description
`callID`	string (50)	Call identifier. See Identifiers and client references [p.11].
`conferenceID`	string (50)	Identifier of the conference to which the call is connected or is in the process of connecting. See Identifiers and client references [p.11].
`conferenceState`	string	State of call connection to the conference. See Table 16: Call conference state enumerated type [p.24].
`callState`	string	State of the call. See Table 17: Call state enumerated type [p.24].
`incoming`	boolean	Direction of the call: **`true`** indicates incoming; **`false`** indicates outgoing.
`protocol`	string	Call control protocol. See Table 12: Call protocol enumerated type [p.23].

Table 62: flex.call.status returned data (continued)

Parameter name	Type	Description
`address`	string (80)	Address of the endpoint. For outgoing calls, this is the destination of the call.
`participantID`	string (50)	Participant identifier. See Identifiers and client references [p.11]. Returned if string length > 0.
`duration`	integer (>= 0)	Duration of the call, in seconds. Only returned if a call has been established.
`rxBandwidth`	integer (>= 0)	Receive bandwidth. Only returned if a call has been established.
`txBandwidth`	integer (>= 0)	Transmit bandwidth. Only returned if a call has been established.
`remoteName`	string (80)	Endpoint name supplied by the far end. Returned if string length > 0.

flex.conference.create

Creates a conference with the supplied parameters and returns the unique identifier of the new conference.

The following parameters must be present for this command to succeed:

- `participantMediaResources.mediaTokensMainVideo.total`
- `participantMediaResources.mediaTokensExtendedVideo.total`
- `participantMediaResources.mediaTokensAudio.total`
- `participantMediaResources.numMediaCredits`

Table 63: `flex.conference.create` inputs

Parameter name	Type	Description
`participantMediaResources`	struct	**Required**. This Participant media resources struct [p.28] defines the conference default participant media resource configuration. These defaults can be overridden by the media resource configuration defined for the conference URI or for the participant.
`conferenceReference`	string (50)	Conference client reference string. See Identifiers and client references [p.11]. Default: empty.
`conferenceName`	string (80)	Human readable label for the conference. Default: empty string.
`conferenceDescription`	string (500)	Human readable description of the conference. If set, the TelePresence Server uses this value for ActiveControl's conference description. Default: empty string.

Table 63: flex.conference.create inputs (continued)

Parameter name	Type	Description
URIs	array of structs	Each member of the array is a Conference URI details struct [p.34] defining a unique conference URI and its associated access levels and media tokens. You may supply a maximum of two conference URI structs. If you want two levels of access to the conference, for example for chairpersons and guests, you must create two structs in this array. Each struct in the array requires a unique URI. Default: empty array.
conferenceMediaTokens	integer (>= 0)	Maximum number of media tokens that can be used for the conference. Do not supply **conferenceMediaTokens** if you supply **conferenceMediaTokensUnlimited**. This will result in a fault. Default: Absent. See **conferenceMediaTokensUnlimited**.
conferenceMediaTokensUnlimited	boolean	Whether no limit is defined for the number of media tokens that can be used for the conference. Do not supply **conferenceMediaTokensUnlimited** if you supply **conferenceMediaTokens**. This will result in a fault. See "Unlimited" integers [p.15]. Default: **true**.
conferenceMediaCredits	integer (>= 0)	Maximum number of media credits that can be used for the conference. Do not supply **conferenceMediaCredits** if you supply **conferenceMediaCreditsUnlimited**. This will result in a fault. Default: Absent. See **conferenceMediaCreditsUnlimited**.
conferenceMediaCreditsUnlimited	boolean	Whether no limit is defined for the number of media credits that can be used for the conference. Do not supply **conferenceMediaCreditsUnlimited** if you supply **conferenceMediaCredits**. This will result in a fault. See "Unlimited" integers [p.15]. Default: **true**.
waitForChair	boolean	Whether callers must wait for a chair to join the conference. Default: **true**.
disconnectOnChairExit	boolean	Whether callers are disconnected when the last chair leaves the conference. Default: **false**.
terminateWithLastCall	boolean	Whether the conference is destroyed with the last call. Default: **false**.
locked	boolean	Whether the conference is locked, causing incoming calls to be rejected. Default: **false**.

Table 63: flex.conference.create inputs (continued)

Parameter name	Type	Description
startTime	integer (>= 0)	Number of seconds to wait before starting the conference. Default: 0.
duration	integer (>= 0)	Conference duration (in seconds) measured from the start time. If the conference duration is due to end in less than 120 seconds, participants are notified that it is about to end, as described in Conference send warning. Do not supply **duration** if you supply **durationUnlimited**. This will result in a fault. Default: Absent. See **durationUnlimited**.
durationUnlimited	boolean	Whether an unlimited duration is assigned for the conference. If this field is present and the value is **true**, the value for the conference duration is ignored. Do not supply **durationUnlimited** if you supply **duration**. This will result in a fault. See "Unlimited" integers [p.15]. Default: **true**.
billingCode	string (80)	Billing code string. Default: empty.
callAttributes	struct	Conference default call attributes. See How call attributes are derived [p.29]. Default: see Table 30: callAttributes struct members [p.30] for defaults of struct members.
maxParticipants	integer (>= 0)	Maximum number of participants that can connect to this conference. Do not supply **maxParticipants** if you supply **maxParticipantsUnlimited**. This will result in a fault. Default: Absent. See **maxParticipantsUnlimited**.
maxParticipantsUnlimited	boolean	Whether an unlimited number of participants are allowed to connect to this conference. Do not supply **maxParticipantsUnlimited** if you supply **maxParticipants**. This will result in a fault.See "Unlimited" integers [p.15]. Default: **true**.
voiceSwitchingSensitivity	integer (0–100)	Voice switching sensitivity. Default: 60.
welcomeScreen	boolean	Whether a welcome screen message is displayed for 5 seconds when a call joins the conference. See **welcomeScreenMessage** for contents of the message. Default: **true**.

Table 63: flex.conference.create inputs (continued)

Parameter name	Type	Description
welcomeScreenMessage	string (500)	Welcome screen message for this conference. If this string is empty, the conference name is displayed as the welcome screen message. If **conferenceDescription** is not set, the TelePresence Server uses this value for ActiveControl's conference description. Default: empty string.
useCustomPINEntryMessage	boolean	Whether a custom message is displayed in the PIN entry screen. Default: **false**.
customPINEntryMessage	string (200)	Custom message for PIN entry. Only used if **useCustomPINEntryMessage** is **true**. Default: empty.
useCustomPINIncorrectMessage	boolean	Whether a custom warning message is displayed in the PIN entry screen after an incorrect PIN has been entered. Default: **false**.
customPINIncorrectMessage	string (100)	Custom warning message for incorrect PIN entry. Only used if **useCustomPINIncorrectMessage** is **true**. Default: empty.
useCustomWaitingForChairMessage	boolean	Whether a custom message is displayed when waiting for a chair to join the conference. Default: **false**.
customWaitingForChairMessage	string (500)	Custom message displayed when waiting for a chair to join the conference. Only used if **useCustomWaitingForChairMessage** is **true**. Default: empty.
use CustomOnlyVideoParticipantMessage	boolean	Whether a custom message is displayed when a participant is the only (active) video participant. Default: **false**.
customOnlyVideoParticipantMessage	string (500)	Custom message displayed when a participant is the only (active) video participant. Only used if **useCustomOnlyVideoParticipantMessage** is **true**. Default: empty.
use CustomConferenceEndingMessage	boolean	Whether a custom message is displayed when the conference is about to end. Default: **false**.
customConferenceEndingMessage	string (100)	Custom message displayed when the conference is about to end. Only used if **useCustomConferenceEndingMessage** is **true**. Default: empty.
metadata	base64 (<= 512 bytes)	Client meta data. Default: zero length.
unlockWithLastCall	boolean	Whether the conference is unlocked when the participant leaves the conference. Default: **true**.

Table 63: flex.conference.create inputs (continued)

Parameter name	Type	Description
guestControlLevel	string	See Table 27: Control level enumerated type [p.27]. Either **controlNone**, **controlLocal**, or **controlConference**. Defines the level of control to which the guests in this conference are entitled. Default: **controlLocal**
chairControlLevel	string	See Table 27: Control level enumerated type [p.27]. Either **controlNone**, **controlLocal**, or **controlConference**. Defines the level of control to which the chairpersons in this conference are entitled. Default: **controlConference**
optimizationProfile	string	Sets the optimization profile for the conference, which defines how the conference reports far-end token values for endpoints.See Table 20: Optimization profiles enumerated type [p.25]. Default: **favorExperience**.
useCustomMutedCanUnmuteMessage	boolean	**true** enables a custom message that will be displayed to participants when their audio input has been muted on the TelePresence Server and they can unmute with *6. Default: **true**.
customMutedCanUnmuteMessage	string (500)	The message displayed to participants when their audio has been muted on the TelePresence Server and they can unmute with *6. Will not be displayed if **useCustomMutedCanUnmuteMessage** is **false**. Default: Empty.
useCustomMutedCannotUnmuteMessage	boolean	**true** enables a custom message that will be displayed when their audio has been muted on the TelePresence Server and they cannot unmute with *6. Default: **true**.
customMutedCannotUnmuteMessage	string (500)	The message displayed to participants when their audio has been muted on the TelePresence Server and they cannot unmute with *6. Will not be displayed if **useCustomMutedCannotUnmuteMessage** is **false**. Default: Empty.

Table 64: **flex.conference.create** returned data

Parameter name	Type	Description
conferenceID	string (50)	Conference identifier assigned by the TelePresence Server. All subsequent invocations of commands to control or query this conference must use this identifier to reference it. See Identifiers and client references [p.11].
conferenceReference	string (50)	Conference client reference string. Returned if string length > 0. See Identifiers and client references [p.11].

flex.conference.deletions.enumerate

Enumerates deleted conferences. The enumeration returns conferences that have been newly deleted.

Table 65: **`flex.conference.deletions.enumerate`** inputs

Parameter name	Type	Description
`cookie`	string (150)	Conference enumeration cookie. This field must be absent when starting an enumeration, and present (using the value returned by a previous invocation) when continuing an enumeration. Default: none.
`max`	integer (> 0)	Maximum number of conference deletion records to return in response. If **`max`** is not specified, as many records are returned as is possible.

Table 66: **`flex.conference.deletions.enumerate`** returned data

Parameter name	Type	Description
`moreAvailable`	boolean	Whether there are more conference deletions to be enumerated.
`cookie`	string (150)	Cookie that must be returned in the next invocation to continue the enumeration.
`conferenceIDs`	array of identifiers	The identifiers of conferences that have been deleted. See Identifiers and client references [p.11].

flex.conference.destroy

Destroys the specified conference. No parameters are returned.

Table 67: **`flex.conference.destroy`** inputs

Parameter name	Type	Description
`conferenceID`	string (50)	**Required**. Conference identifier assigned by the TelePresence Server. See Identifiers and client references [p.11].

flex.conference.enumerate

Enumerates conferences controlled by the TelePresence Server. The enumeration returns new conferences as well as conferences that have changed since the last invocation of the enumeration command. See Enumeration [p.16].

It is possible for two or more instances in succession of enumeration information structs returned for a particular conference to be identical. This may happen in the following circumstances:

- The summed token values are the same as before, but the distribution of tokens across participants has changed.
- Distribution of tokens was different for some period of time between successive invocations of the flex.conference.enumerate command.

Table 68: **`flex.conference.enumerate`** inputs

Parameter name	Type	Description
`cookie`	string (150)	Conference enumeration cookie. This field must be absent when starting an enumeration, and present (using the value returned by a previous invocation) when continuing an enumeration. Default: none.
`max`	integer (> 0)	Maximum number of conference details to return in response. If **`max`** is not specified, as many records will be returned as is possible.

Table 69: **`flex.conference.enumerate`** returned data

Parameter name	Type	Description
`moreAvailable`	boolean	Whether there are more conferences to be enumerated.
`cookie`	string (150)	Cookie that must be returned in the next invocation to continue the enumeration.
`conferences`	array of structs	Each member of the array is a struct representing a single conference. See Table 70: Conference information struct [p.55]. The array is returned empty if there are no conferences to enumerate.

The enumerated conference information struct contains two sets of media resource values (tokens and credits).

- Configured: values set by configuration typically using this API.
- Allocated: the minimum of resource values corresponding to the capabilities of the near end and far end if a connection to the far end exists, *or* the configured values if media resources have been reserved and a call has not been established.

Table 70: Conference information struct

Parameter name	Type	Description
`conferenceID`	string (50)	Conference identifier assigned by the TelePresence Server. See Identifiers and client references [p.11].
`conferenceReference`	string (50)	Conference client reference string. Returned if string length > 0. See Identifiers and client references [p.11].
`conferenceName`	string (80)	Human readable label for the conference. Returned if string length > 0.
`locked`	boolean	Whether the conference is locked.
`active`	boolean	Whether the conference has started.
`numParticipants`	integer (>= 0)	Number of participants connected to the conference, including participants with incoming calls that have not yet been established.
`callTokensConfiguredMainVideo`	integer (>= 0)	Number of tokens configured for main video summed over all participants connected to the conference.

Table 70: Conference information struct (continued)

Parameter name	Type	Description
`callTokensAllocatedMainVideo`	integer (>= 0)	Number of tokens allocated for main video summed over all participants connected to the conference.
`callTokensConfiguredExtendedVideo`	integer (>= 0)	Number of tokens configured for extended video summed over all participants connected to the conference.
`callTokensAllocatedExtendedVideo`	integer (>= 0)	Number of tokens allocated for extended video summed over all participants connected to the conference.
`callTokensConfiguredAudio`	integer (>= 0)	Number of tokens configured for audio summed over all participants connected to the conference.
`callTokensAllocatedAudio`	integer (>= 0)	Number of tokens allocated for audio summed over all participants connected to the conference.
`callQualityCanImproveMainVideo`	boolean	Whether at least one participant exists for which the number of far end main video tokens exceeds the number of configured main video tokens. The quality of the call can be improved by increasing the number of configured tokens.
`callQualityCanImproveExtendedVideo`	boolean	Whether at least one participant exists for which the number of far end extended video tokens exceeds the number of configured extended video tokens. The quality of the connection can be improved by increasing the number of configured tokens.
`callQualityCanImproveAudio`	boolean	Whether at least one participant exists for which the number of far end audio tokens exceeds the number of configured audio tokens. The quality of the connection can be improved by increasing the number of configured tokens.
`creditsConfigured`	integer (>= 0)	Number of configured credits summed over all participants connected to the conference.
`creditsAllocated`	integer (>= 0)	Number of allocated credits summed over all participants connected to the conference.
`presenterID`	string (50)	The identifier of the participant who is currently presenting to the conference. Not returned if no participant is presenting.
`importantID`	string (50)	The identifier of the participant who is currently marked as the conference's important participant. Not returned if no participant is marked as important. See flex.participant.setImportant [p.88] and flex.participant.clearImportant [p.70].

flex.conference.getMetadata

Returns metadata associated with the specified conference. If a conference does not have metadata, 0 length metadata is returned.

Table 71: **`flex.conference.getMetadata`** inputs

Parameter name	Type	Description
`conferenceID`	string (50)	**Required**. Conference identifier assigned by the TelePresence Server. See Identifiers and client references [p.11].

Table 72: **`flex.conference.getMetadata`** returned data

Parameter name	Type	Description
`metadata`	base64 (<= 512 bytes)	Client meta data.

flex.conference.modify

Updates the specified parameters of the specified conference. No parameters are returned.

Settings that you do not specify remain as they were, unless implicitly affected by other settings that you supply. For example, if **`duration`** is specified, **`durationUnlimited`** is implicitly set to **`false`**.

The following pairs of parameters must be specified in compliance with the requirements stated in "Unlimited" integers [p.15]:

- **`duration`** and **`durationUnlimited`**
- **`maxParticipants`** and **`maxParticipantsUnlimited`**
- **`conferenceMediaTokens`** and **`conferenceMediaTokensUnlimited`**
- **`conferenceMediaCredits`** and **`conferenceMediaCreditsUnlimited`**

Table 73: **`flex.conference.modify`** inputs

Parameter name	Type	Description
`conferenceID`	string (50)	**Required**. Conference identifier assigned by the TelePresence Server. See Identifiers and client references [p.11].
`conferenceReference`	string (50)	Conference client reference string. See Identifiers and client references [p.11].
`conferenceName`	string (80)	Human readable label for the conference.
`conferenceDescription`	string (500)	Human readable description of the conference. If set, the TelePresence Server uses this value for ActiveControl's conference description.

Table 73: flex.conference.modify inputs (continued)

Parameter name	Type	Description
URIs	array of structs	Each member of the array is a Conference URI details struct [p.34] that defines conference URIs, associated access levels, and media resources. These replace all earlier conference URI entries. For URIs specified, all fields are set; unspecified optional fields are set to default values. If you supply an empty array, any previously defined conference URIs are deleted. You may supply a maximum of two conference URI structs. If you want two levels of access to the conference, for example for chairpersons and guests, you must create two structs in this array. Each struct in the array requires a unique URI.
conferenceMediaTokens	integer (>= 0)	Maximum number of media tokens that can be used for the conference. See **conferenceMediaTokensUnlimited**.
conferenceMediaTokensUnlimited	boolean	Whether no limit is defined for the number of media tokens that can be used for the conference.
conferenceMediaCredits	integer (>= 0)	Maximum number of media credits that can be used for the conference. See **conferenceMediaCreditsUnlimited**.
conferenceMediaCreditsUnlimited	boolean	Whether no limit is defined for the number of media credits that can be used for the conference.
waitForChair	boolean	Whether callers must wait for a chair to join the conference.
disconnectOnChairExit	boolean	Whether callers are disconnected when the last chair leaves the conference.
terminateWithLastCall	boolean	Whether the conference is destroyed with the last call.
locked	boolean	Whether the conference is locked causing incoming calls to be rejected.
duration	integer (>= 0)	Conference duration (in seconds) measured from the start time. If the conference is due to end in less than 120 seconds, participants are notified that it is about to end, as described in Conference send warning. It is an error to set the duration to a value that requires it to have ended in the past. See **durationUnlimited**.
durationUnlimited	boolean	Whether an unlimited duration is assigned for the conference.
billingCode	string (80)	Billing code string.
callAttributes	struct	This Call attributes struct [p.29] defines the conference default call attributes. See How call attributes are derived [p.29].

Table 73: flex.conference.modify inputs (continued)

Parameter name	Type	Description
participantMediaResources	struct	This Participant media resources struct [p.28] struct defines the conference default participant media resource configuration. These settings can be overridden by those defined at the **Conference URI** or participant level. All members of the struct are changed; unspecified optional fields are set to their default values.
maxParticipants	integer (>= 0)	Maximum number of participants that can connect to this conference. See **maxParticipantsUnlimited**.
maxParticipantsUnlimited	boolean	Whether an unlimited number of participants are allowed to connect to this conference. See "Unlimited" integers [p.15].
voiceSwitchingSensitivity	integer (0–100)	Voice switching sensitivity. Default: 60.
welcomeScreen	boolean	Whether a welcome screen message is displayed for 5 seconds when a call joins the conference. See **welcomeScreenMessage** for the contents of the message.
welcomeScreenMessage	string (500)	Welcome screen message for this conference. If this message is empty, the conference name is displayed as the welcome screen message. If **conferenceDescription** is not set, the TelePresence Server uses this value for ActiveControl's conference description.
useCustomPINEntryMessage	boolean	Whether a custom message is displayed in the PIN entry screen.
customPINEntryMessage	string (200)	Custom message for PIN entry. Only used if **useCustomPINEntryMessage** is **true**.
useCustomPINIncorrectMessage	boolean	Whether a custom warning message is displayed in the PIN entry screen after an incorrect PIN has been entered.
customPINIncorrectMessage	string (100)	Custom warning message for incorrect PIN entry. Only used if **useCustomPINIncorrectMessage** is **true**.
useCustomWaitingForChairMessage	boolean	Whether a custom message is displayed when waiting for a chair to join the conference.
customWaitingForChairMessage	string (500)	Custom message displayed when waiting for a chair to join the conference. Only used if **useCustomWaitingForChairMessage** is **true**.
use CustomOnlyVideoParticipantMessage	boolean	Whether a custom message is displayed when a participant is the only (active) video participant.
customOnlyVideoParticipantMessage	string (500)	Custom message displayed when a participant is the only (active) video participant. Only used if **useCustomOnlyVideoParticipantMessage** is **true**.

Table 73: flex.conference.modify inputs (continued)

Parameter name	Type	Description
useCustomConferenceEndingMessage	boolean	Whether a custom message is displayed when the conference is about to end.
customConferenceEndingMessage	string (100)	Custom message displayed when the conference is about to end. Only used if **useCustomConferenceEndingMessage** is **true**.
metadata	base64 (<= 512 bytes)	Client metadata.
unlockWithLastCall	boolean	Whether the conference is unlocked when the participant leaves the conference.
guestControlLevel	string	See Table 27: Control level enumerated type [p.27]. Either **controlNone**, **controlLocal**, or **controlConference**. Defines the level of control to which the guests in this conference are entitled.
chairControlLevel	string	See Table 27: Control level enumerated type [p.27]. Either **controlNone**, **controlLocal**, or **controlConference**. Defines the level of control to which the chairpersons in this conference are entitled.
optimizationProfile	string	Sets the optimization profile for the conference, which defines how the conference reports far-end token values for endpoints.See Table 20: Optimization profiles enumerated type [p.25]. Default: **favorExperience**.
useCustomMutedCanUnmuteMessage	boolean	**true** enables a custom message that will be displayed to participants when their audio input has been muted on the TelePresence Server and they can unmute with *6. Default: **true**.
customMutedCanUnmuteMessage	string (500)	The message displayed to participants when their audio has been muted on the TelePresence Server and they can unmute with *6. Will not be displayed if **useCustomMutedCanUnmuteMessage** is **false**. Default: Empty.
useCustomMutedCannotUnmuteMessage	boolean	**true** enables a custom message that will be displayed when their audio has been muted on the TelePresence Server and they cannot unmute with *6. Default: **true**.
customMutedCannotUnmuteMessage	string (500)	The message displayed to participants when their audio has been muted on the TelePresence Server and they cannot unmute with *6. Will not be displayed if **useCustomMutedCannotUnmuteMessage** is **false**. Default: Empty.

flex.conference.query

Returns the parameters of a specified conference.

Table 74: `flex.conference.query` inputs

Parameter name	Type	Description
`conferenceID`	string (50)	**Required**. Conference identifier assigned by the TelePresence Server. See Identifiers and client references [p.11].

Table 75: `flex.conference.query` returned data

Parameter name	Type	Description
`conferenceID`	string (50)	Conference identifier assigned by the TelePresence Server. See Identifiers and client references [p.11].
`URIs`	array of structs	Each member of the array is a Conference URI details struct [p.34] defining a unique conference URI and its associated access levels and media tokens. The array contains a maximum of two conference URI structs. These can be used to allow two levels of access to the conference, for example for chairpersons and guests. Each struct in the array requires a unique URI.
`conferenceReference`	string (50)	Conference client reference string. Returned if string length > 0. See Identifiers and client references [p.11].
`conferenceName`	string (80)	Human readable label for the conference. Returned if string length > 0.
`conferenceDescription`	string (500)	Human readable description of the conference. If set, the TelePresence Server uses this value for ActiveControl's conference description. Not returned if it has not been set.
`conferenceMediaTokens`	integer (>= 0)	Maximum number of media tokens that can be used for the conference. See `conferenceMediaTokensUnlimited`. Not returned if it has not been set.
`conferenceMediaTokensUnlimited`	boolean	Whether no limit is defined for the number of media tokens that can be used for the conference. Not returned if it has not been set. See "Unlimited" integers [p.15].
`conferenceMediaCredits`	integer (>= 0)	Maximum number of media credits that can be used for the conference. See `conferenceMediaCreditsUnlimited`. Not returned if it has not been set.

Table 75: flex.conference.query returned data (continued)

Parameter name	Type	Description
conferenceMediaCreditsUnlimited	boolean	Whether no limit is defined for the number of media credits that can be used for the conference. Not returned if it has not been set. See "Unlimited" integers [p.15].
duration	integer (>= 0)	Conference duration (in seconds) measured from the start time. If the conference is due to end in less than 120 seconds, participants are notified that it is about to end, as described in Conference send warning. Not returned if it has not been set. See **durationUnlimited**.
durationUnlimited	boolean	Whether an unlimited duration is assigned for the conference. Not returned if it has not been set. See "Unlimited" integers [p.15].
billingCode	string (80)	Billing code string. Not returned if it has not been set.
maxParticipants	integer (>= 0)	Maximum number of participants that can connect to this conference. Not returned if it has not been set. See **maxParticipantsUnlimited**.
maxParticipantsUnlimited	boolean	Whether an unlimited number of participants are allowed to connect to this conference. Not returned if it has not been set. See "Unlimited" integers [p.15].
waitForChair	boolean	Whether callers must wait for a chair to join the conference.
disconnectOnChairExit	boolean	Whether callers are disconnected when the last chair leaves the conference.
terminateWithLastCall	boolean	Whether the conference is destroyed with the last call.
locked	boolean	Whether the conference is locked so that only outgoing calls are permitted.
startTime	integer (>= 0)	Number of seconds after which to start the conference.
callAttributes	struct	This Call attributes struct [p.29] defines the conference default call attributes. See How call attributes are derived [p.29].
voiceSwitchingSensitivity	integer (0–100)	Voice switching sensitivity.
participantMediaResources	struct	This Participant media resources struct [p.28] defines the conference default participant media resource configuration. These settings can be over-ridden by those defined at the **Conference URI** or participant level.

Table 75: flex.conference.query returned data (continued)

Parameter name	Type	Description
`welcomeScreen`	boolean	Whether a welcome screen message is displayed for 5 seconds when a call joins the conference. See **`welcomeScreenMessage`** for contents of the message.
`welcomeScreenMessage`	string (500)	Welcome screen message for this conference. If this message is empty, the conference name is displayed as the welcome screen message. If **`conferenceDescription`** is not set, the TelePresence Server uses this value for ActiveControl's conference description.
`useCustomPINEntryMessage`	boolean	Whether a custom message is displayed in the PIN entry screen.
`customPINEntryMessage`	string (200)	Custom message for PIN entry. Only used if **`useCustomPINEntryMessage`** is **`true`**.
`useCustomPINIncorrectMessage`	boolean	Whether a custom warning message is displayed in the PIN entry screen after an incorrect PIN has been entered.
`customPINIncorrectMessage`	string (100)	Custom warning message for incorrect PIN entry, Only used if **`useCustomPINIncorrectMessage`** is **`true`**.
`useCustomWaitingForChairMessage`	boolean	Whether a custom message is displayed when waiting for a chair to join the conference.
`customWaitingForChairMessage`	string (500)	Custom message displayed when waiting for a chair to join the conference. Only used if **`useCustomWaitingForChairMessage`** is **`true`**.
`useCustomOnlyVideoParticipantMessage`	boolean	Whether a custom message is displayed when the participant is the only (active) video participant.
`customOnlyVideoParticipantMessage`	string (500)	Custom message displayed when the participant is the only (active) video participant. Only used if **`useCustomOnlyVideoParticipantMessage`** is **`true`**.
`useCustomConferenceEndingMessage`	boolean	Whether a custom message is displayed when the conference is about to end.
`customConferenceEndingMessage`	string (100)	Custom message displayed when the conference is about to end. Only used if **`useCustomConferenceEndingMessage`** is **`true`**.
`hasMetadata`	boolean	Whether the conference has non-zero-length metadata.
`unlockWithLastCall`	boolean	Whether the conference is unlocked when the last participant leaves the conference.

Table 75: flex.conference.query returned data (continued)

Parameter name	Type	Description
guestControlLevel	string	See Table 27: Control level enumerated type [p.27]. Either **controlNone**, **controlLocal**, or **controlConference**. Defines the level of control to which the guests in this conference are entitled.
chairControlLevel	string	See Table 27: Control level enumerated type [p.27]. Either **controlNone**, **controlLocal**, or **controlConference**. Defines the level of control to which the chairpersons in this conference are entitled.
optimizationProfile	string	The optimization profile for the conference, which defines how the conference reports far-end token values for endpoints.See Table 20: Optimization profiles enumerated type [p.25].
useCustomMutedCanUnmuteMessage	boolean	**true** means that a custom message can be displayed to participants when their audio input has been muted on the TelePresence Server and they can unmute with *6.
customMutedCanUnmuteMessage	string (500)	The message that will be displayed to participants when their audio input has been muted on the TelePresence Server and they can unmute with *6. Will not be displayed if **useCustomMutedCanUnmuteMessage** is **false**.
useCustomMutedCannotUnmuteMessage	boolean	**true** means that a custom message can be displayed to participants when their audio input has been muted on the TelePresence Server and they cannot unmute with *6.
customMutedCannotUnmuteMessage	string (500)	The message that will be displayed to participants when their audio input has been muted on the TelePresence Server and they cannot unmute with *6. Will not be displayed if **useCustomMutedCannotUnmuteMessage** is **false**.

flex.conference.sendUserMessage

Sends a message to all participants in the conference. For multi-call participants, the message is sent to the call in the center.

Table 76: **flex.conference.sendUserMessage** inputs

Parameter name	Type	Description
conferenceID	string (50)	**Required**. Conference identifier assigned by the TelePresence Server. See Identifiers and client references [p.11].
message	string (500)	**Required**. Message to display.

Table 76: flex.conference.sendUserMessage inputs (continued)

Parameter name	Type	Description
`position`	integer (1–9)	Position on the display: | 1 2 3 | | 4 5 6 | | 7 8 9 | Default: 5.
`duration`	integer (>0)	Duration in seconds for the message display on the endpoint. The TelePresence Server will accept `0` but the behavior is undefined in this case. Default: 30 seconds.

flex.conference.sendWarning

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

If possible, a participant is notified that the conference is about to end using an appropriate out-of-band protocol. Otherwise, a message is rendered on the participant screen.

Table 77: **`flex.conference.sendWarning`** inputs

Parameter name	Type	Description
`conferenceID`	string (50)	**Required**. Conference identifier assigned by the TelePresence Server. See Identifiers and client references [p.11].
`secondsRemaining`	integer (>=0)	Additional information for the warning, namely the amount of time remaining until the conference is expected to terminate. Some endpoints are capable of receiving and using this information. Setting this value will **not** result in termination of the conference after the specified amount of time. Default: 120 seconds if the conference has no defined ending time. There is no default for conferences with a finite duration.

flex.conference.status

Returns the status of the specified conference.

Table 78: **`flex.conference.status`** inputs

Parameter name	Type	Description
`conferenceID`	string (50)	**Required**. Conference identifier assigned by the TelePresence Server. See Identifiers and client references [p.11].

Table 79: **`flex.conference.status`** returned data

Parameter name	Type	Description
`conferenceID`	string (50)	Conference identifier assigned by the TelePresence Server. See Identifiers and client references [p.11].

Table 79: flex.conference.status returned data (continued)

Parameter name	Type	Description
locked	boolean	Whether the conference is locked.
numParticipants	integer (>= 0)	Number of participants connected to the conference, including participants with incoming calls that have not yet been established.
numScreens	integer (>= 0)	Number of screens connected to the conference.
configuredMediaTokens	integer	Sum of configured media tokens for all calls connected to the conference.
configuredMediaCredits	integer	Sum of configured media credits for all calls connected to the conference.
startTime	integer	Amount of time, measured in seconds, between now and the conference start time. This value is < 0 for conferences that are currently active.
active	boolean	Whether the conference has started.
presenterID	string (50)	The identifier of the participant who is currently presenting to the conference. Not returned if no participant is presenting.
conferenceReference	string (50)	Conference client reference string. Returned if string length > 0. See Identifiers and client references [p.11].
endTime	integer	Amount of time, measured in seconds, until the conference is due to end. Only returned if conference duration is limited.

flex.participant.advanced.enumerate

Enumerates participants. This command is an alternative for **`flex.participant.enumerate`**. **`flex.participant.enumerate`** is still accepted, but you should only use one of these methods for participant enumeration.

See Enumeration [p.16] and flex.participant.enumerate [p.73].

Table 80: **`flex.participant.advanced.enumerate`** inputs

Parameter name	Type	Description
cookie	string (150)	Participant enumeration cookie. This field must be absent at the start of the enumeration, and present (using the value returned by a previous invocation) when continuing an enumeration. Default: none.
max	integer (> 0)	Maximum number of participant details to return in response. If **max** is not specified, as many records will be returned as is possible.
conferenceID	string (50)	Enumerates only participants in the specified conference. Can only be supplied when **cookie** is absent. Enumeration for non-existent conferences will fail. See Identifiers and client references [p.11]. Default: none.

Table 81: **`flex.participant.advanced.enumerate`** returned data

Parameter name	Type	Description
`moreAvailable`	boolean	Whether there are more participants to be enumerated.
`cookie`	string (150)	Enumeration cookie that must be returned in the next invocation to continue the enumeration. See Enumeration [p.16].
`participants`	array of structs	Each member of the array is a struct defining a participant. See Table 82: Participant information struct members [p.67]. This array may be empty.

Table 82: Participant information struct members

Parameter name	Type	Description
`participantID`	string (50)	Participant identifier assigned by the TelePresence Server. See Identifiers and client references [p.11].
`participantReference`	string (50)	Client reference string. See Identifiers and client references [p.11]. Absent if empty.
`displayName`	string (80)	The current display name for this participant. Absent if empty.
`conferenceID`	string (50)	Conference to which the participant is connected. See Identifiers and client references [p.11].
`accessLevel`	string	Access level granted to the participant. See Table 7: Access level enumerated type [p.22].
`connectionState`	string	One of **`disconnected`**, **`connecting`**, **`connected`**, or **`onHold`**. See Table 22: Participant connection state enumerated type [p.25].
`calls`	array of structs	Each member of this array is a struct that defines one call associated with the participant. See Table 83: Participant call information struct members [p.68]. If a call is absent, the array member is empty. The array position of the call information struct matches the position of the corresponding address struct in the **`addresses`** array.
`addresses`	array of structs	Each member of this array is a struct that contains either the address or the URI for one of the calls associated with the participant. See Table 84: Participant address struct members [p.68]. The position in the array matches that of the associated call in the **`calls`** array.
`encryptionStatus`	struct	An overview of the participant's encryption status. Each struct member represents a channel, or category of channels, that can be encrypted for this participant. The value of each member is one of the encryption status enumerated types. See Table 85: encryptionStatus struct members [p.69] and Table 23: Encryption status enumerated type [p.26].

Table 82: Participant information struct members (continued)

Parameter name	Type	Description
layout	string	The display layout that is currently shown on the participant's endpoint. One of **layoutSingle**, **layoutActivepresence**, **layoutProminent**, or **layoutEqual**. Not returned if there are no calls connected for this participant. See Table 10: Single screen layout enumerated type [p.23] or Table 11: Multi-screen layout enumerated type [p.23], depending on the type of endpoint.
mediaStatus	struct	An overview of the participant's media status. Each struct member represents a media channel, or category of media channels, that can be negotiated for this participant. The value of each member is one of the media status enumerated types. See Table 86: mediaStatus struct members [p.69] and Table 24: Media status enumerated type [p.26].

Table 83: Participant call information struct members

Parameter name	Type	Description
callID	string (50)	Call identifier assigned by the TelePresence Server. See Identifiers and client references [p.11].
incoming	boolean	Direction of the call: **true** indicates incoming; **false** indicates outgoing.
address	string (80)	Address of the endpoint. For outgoing calls, this is the destination of the call.
protocol	string	The call control protocol used in this call. One of **h323** or **sip**. See Table 18: Call protocol enumerated type [p.24].
rxBandwidth	integer	
txBandwidth	integer	

Table 84: Participant address struct members

Parameter name	Type	Description
URI	string (80)	URI used or to be used by the endpoint to connect to the conference. Incoming calls only, not returned for outgoing calls. Only returned if it is not empty.
remoteAddress	string (80)	Remote address specified in flex.participant.create [p.70] to call out to the endpoint. Outgoing calls only, not returned for incoming calls. Only returned if it is not empty.

Table 85: **`encryptionStatus`** struct members

Parameter name	Type	Description
`callProtocol`	string	The encryption status of the call control protocol. One of **`unknown`**, **`encrypted`**, **`mixed`**, or **`unencrypted`**. See Table 23: Encryption status enumerated type [p.26].
`audio`	string	The encryption status of this participant's audio channel(s)
`mainVideo`	string	The encryption status of this participant's video channel(s)
`extendedVideo`	string	The encryption status of this participant's content channel
`cccp`	string	The encryption status of the Cisco Conference Control Protocol (CCCP).
`activeControl`	string	The encryption status of the ActiveControl signaling channel.

Table 86: **`mediaStatus`** struct members

Parameter name	Type	Description
`audioRx`	string	Media status of the audio channel(s) received from this participant. One of **`notNegotiated`**, **`inUse`**, **`notInUse`**, or **`muted`**. See Table 24: Media status enumerated type [p.26].
`audioTx`	string	Media status of the audio channel(s) transmitted to this participant.
`videoRx`	string	Media status of the video channel(s) received from this participant.
`videoTx`	string	Media status of the video channel(s) transmitted to this participant.
`extendedRx`	string	Media status of the content channel(s) received from this participant.
`extendedTx`	string	Media status of the content channel(s) transmitted to this participant.

flex.participant.call.disconnect

Disconnects an incoming call that is connected through a participant conference URI.

Outgoing calls cannot be disconnected. To change the destination of an outgoing call, the participant must be destroyed and recreated with the new address.

Table 87: **`flex.participant.call.disconnect`** inputs

Parameter name	Type	Description
`participantID`	string (50)	**Required**. Participant identifier assigned by the TelePresence Server. See Identifiers and client references [p.11].
`position`	integer (>= 0)	**Required**. Relative position of the endpoint in the group. 0 represents the leftmost screen from a viewing position.

This command fails under the following circumstances:

- The participant call has not connected through a participant conference URI.
- The participant call is an outgoing call.
- The position value is invalid for the participant.

flex.participant.clearImportant

Removes the designation of the specified participant as the important participant.

Table 88: **`flex.participant.clearImportant`** inputs

Parameter name	Type	Description
`participantID`	string (50)	**Required**. Participant identifier assigned by the TelePresence Server. See Identifiers and client references [p.11].

flex.participant.create

Creates single- or multi-call participants associated with the specified conference.

If the command is successful, media resources (tokens and credits) are reserved for the new participant. See Media reservation [p.16])

Note: The token requirements for a call cannot be known prior to instantiation of the call, so no checks are made on **`flex.participant.create`** or **`flex.participant.modify`** to determine if the call will have adequate resources. The client is therefore responsible for ensuring that the call has adequate resources.

The following circumstances can cause this command to fail:

- The **`audioIndex`** and **`contentIndex`** values are invalid
- The rules for Participant call definition struct [p.35] are not met
- **`cascadeRole`** is not **`cascadeNone`** and more than one call is defined in the **`calls`** array

Table 89: **`flex.participant.create`** inputs

Parameter name	Type	Description
`conferenceID`	string (50)	**Required**. Conference identifier. See Identifiers and client references [p.11].
`calls`	array of structs	**Required**. Each member of the array is a Participant call definition struct [p.35] which specifes a call for this participant; you must supply a minimum of 1 call definition struct and may supply up to 4. For multi-call participants, the array position of the struct corresponds to the call's physical location with respect to other calls. For example, the array positions of a three-call participant correspond to physical locations as follows: position 0 is left, 1 is center, and 2 is right. For endpoints that automatically negotiate extra screens (such as a T3), you need only specify the call at position 0 and the remaining calls will be added as they are negotiated.

Table 89: flex.participant.create inputs (continued)

Parameter name	Type	Description
`participantReference`	string (50)	Client reference string. See Identifiers and client references [p.11]. Default: empty.
`PIN`	string (40)	Numeric PIN this participant will use when connecting to conference URIs. Participants only need to supply a PIN when calling in to a PIN-protected URI. A PIN is never requested when the TelePresence Server calls out to an endpoint. If a PIN is supplied to this call when it is not required (because all the calls are outgoing), then the TelePresence Server returns fault code 102. Default: empty.
`callAttributes`	struct	See Call attributes struct [p.29] for details of struct members. The settings defined in this struct override the conference's default call attribute settings. See How call attributes are derived [p.29]. Default: inherits conference default call attributes.
`participantMediaResources`	struct	See Participant media resources struct [p.28] for details of struct members. The settings defined in this struct override the conference's default participant media resource configuration. Default: inherits conference default participant media resource configuration.
`camerasCrossed`	boolean	Whether cameras in the group of endpoints specified by **`calls`** are crossed. Ignored if the calls array length = 1. Default: **`false`**.
`audioIndex`	integer (>= 0)	Position in the **`calls`** array of the call that will receive audio. The position must exist in the **`calls`** array. First position is 0. Ignored if the **`calls`** array length = 1. Default: 0.
`contentIndex`	integer (>= 0)	Position in the **`calls`** array of the call that will receive content. The position must exist in the **`calls`** array. First position is 0. Ignored if the **`calls`** array length = 1. Default: 0.
`displayName`	string (80)	Configured display name for the endpoint. This overrides the endpoint display name setting. Default: empty.
`dtmf`	string (127)	Valid DTMF [p.19] characters in a sequence that is sent to the call nominated by the **`audioIndex`**. This sequence is only sent when dialing out and is not sent for incoming calls. Default: empty.
`callerName`	string (80)	Calling name seen by the endpoint. Not used for incoming calls. Default: empty.
`callerAddress`	string (80)	Calling address seen by the endpoint. Not used for incoming calls. Default: empty.
`cascadeRole`	string	One of **`cascadeNone`**, **`cascadeMaster`**, or **`cascadeSlave`**. See Table 19: Cascade roles enumerated type [p.25]. Default: **`cascadeNone`**.

Table 90: `flex.participant.create` returned data

Parameter name	Type	Description
`participantID`	string (50)	Participant identifier assigned by the TelePresence Server. All subsequent invocations of commands to control or query this participant must use this identifier to reference it. See Identifiers and client references [p.11].
`participantReference`	string (50)	Client reference string. Returned if not empty. See Identifiers and client references [p.11].

flex.participant.deletions.enumerate

Enumerates only deleted participants.

The response will include either the `participantIDs` array or the `IDs` array, depending on the value of `extended` that you supply in the first invocation.

Table 91: `flex.participant.deletions.enumerate` inputs

Parameter name	Type	Description
`cookie`	string (150)	Participant deletions enumeration cookie. This field must be absent at the start of the enumeration, and present (using the value returned by a previous invocation) to continue the enumeration. Default: none.
`max`	integer (> 0)	Maximum number of participant deletion records returned in response. If `max` is not specified, as many records are returned as is possible.
`conferenceID`	string (50)	Enumerates only participants in the specified conference. Can only be supplied when `cookie` is absent. Enumeration for non-existent conferences will fail. See Identifiers and client references [p.11]. Default: none.
`extended`	boolean	If `true`, the response includes the `IDs` array. If `false`, the response includes the `participantIDs` array. `extended` is only accepted on the first enumerate command, and is ignored on subsequent enumerations. You cannot change the type of array returned during an enumeration. Default: `false`.

Table 92: `flex.participant.deletions.enumerate` returned data

Parameter name	Type	Description
`moreAvailable`	boolean	Whether there are more participant deletions to be enumerated.
`cookie`	string (150)	Cookie that must be supplied in the next invocation to continue the enumeration.
`participantIDs`	array of strings	Each member of the array is a string (50) that identifies a participant that has been deleted. See Identifiers and client references [p.11]. Not returned if `IDs` is returned.
`IDs`	array of structs	Each member of the array is a struct that identifies a participant that has been deleted, and the conference from which that participant was deleted. See Table 93: IDs array struct members [p.73]. Not returned if `participantIDs` is returned.

Table 93: **`IDs`** array struct members

Parameter name	Type	Description
`participantID`	string (50)	Participant identifier assigned by the TelePresence Server. See Identifiers and client references [p.11].
`conferenceID`	string (50)	Conference identifier assigned by the TelePresence Server. See Identifiers and client references [p.11].

flex.participant.destroy

Destroys the specified participant. Any existing calls are destroyed.

Table 94: **`flex.participant.destroy`** inputs

Parameter name	Type	Description
`participantID`	string (50)	**Required**. Participant identifier assigned by the TelePresence Server. See Identifiers and client references [p.11].

flex.participant.enumerate

Enumerates participants. The alternative command **`flex.participant.advanced.enumerate`** can be used instead, but you should only use one type of participant enumeration.

See Enumeration [p.16] and flex.participant.advanced.enumerate [p.66]

Table 95: **`flex.participant.enumerate`** inputs

Parameter name	Type	Description
`cookie`	string (150)	Participant enumeration cookie. This field must be absent at the start of the enumeration, and present (using the value returned by a previous invocation) when continuing an enumeration. Default: none.
`max`	integer (> 0)	Maximum number of participant details to return in response. If **`max`** is not specified, as many records will be returned as is possible.
`conferenceID`	string (50)	Enumerates only participants in the specified conference. Can only be supplied when **`cookie`** is absent. Enumeration for non-existent conferences will fail. See Identifiers and client references [p.11]. Default: none.

Table 96: **`flex.participant.enumerate`** returned data

Parameter name	Type	Description
`moreAvailable`	boolean	Whether there are more participants to be enumerated.
`cookie`	string (150)	Enumeration cookie that must be returned in the next invocation to continue the enumeration. See Enumeration [p.16].
`participants`	array of structs	Each member of the array is a struct defining a participant. See Table 97: Participant information struct members [p.74]. This array may be empty.

Table 97: Participant information struct members

Parameter name	Type	Description
participantID	string (50)	Participant identifier assigned by the TelePresence Server. See Identifiers and client references [p.11].
participantReference	string (50)	Client reference string. See Identifiers and client references [p.11]. Absent if empty.
displayName	string (80)	The current display name for this participant. Absent if empty.
conferenceID	string (50)	Conference to which the participant is connected. See Identifiers and client references [p.11].
accessLevel	string	Access level granted to the participant. See Table 7: Access level enumerated type [p.22].
connectionState	string	One of **disconnected**, **connecting**, **connected**, or **onHold**. See Table 22: Participant connection state enumerated type [p.25].
calls	array of structs	Each member of this array is a struct that defines one call associated with the participant. See Table 98: Participant call information struct members [p.74]. If a call is absent, the array member is empty. The array position of the call information struct matches the position of the corresponding address struct in the **addresses** array.
addresses	array of structs	Each member of this array is a struct that contains either the address or the URI for one of the calls associated with the participant. See Table 99: Participant address struct members [p.74]. The position in the array matches that of the associated call in the **calls** array.

Table 98: Participant call information struct members

Parameter name	Type	Description
callID	string (50)	Call identifier assigned by the TelePresence Server. See Identifiers and client references [p.11].
incoming	boolean	Direction of the call: **true** indicates incoming; **false** indicates outgoing.
address	string (80)	Address of the endpoint. For outgoing calls, this is the destination of the call.

Table 99: Participant address struct members

Parameter name	Type	Description
URI	string (80)	URI used or to be used by the endpoint to connect to the conference. Incoming calls only, not returned for outgoing calls. Only returned if it is not empty.

Table 99: Participant address struct members (continued)

Parameter name	Type	Description
`remoteAddress`	string (80)	Remote address specified in flex.participant.create [p.70] to call out to the endpoint. Outgoing calls only, not returned for incoming calls. Only returned if it is not empty.

flex.participant.media.enumerate

Enumerates participants for media information. A participant can consist of one or more calls.

The enumeration returns participants that have been newly added and calls that have had changes to token settings since the previous invocation of the method, as indicated by the cookie.

Note: Only one of the parameters **`maxTokensPerChannelConfigured`** and **`maxTokensPerChannelConfiguredUnlimited`** can be returned. See "Unlimited" integers [p.15].

Table 100: **`flex.participant.media.enumerate`** inputs

Parameter name	Type	Description
`cookie`	string (150)	Participant media enumeration cookie. This field must be absent at the start of the enumeration, and present (using the value returned by a previous invocation) when continuing an enumeration. Default: none.
`max`	integer (> 0)	Maximum number of participant media details to return in response. If **`max`** is not specified, as many records will be returned as is possible.
`conferenceID`	string (50)	Enumerates only participants in the specified conference. Can only be supplied when **`cookie`** is absent. Enumeration for non-existent conferences will fail. See Identifiers and client references [p.11]. Default: none.

Table 101: **`flex.participant.media.enumerate`** returned data

Parameter name	Type	Description
`moreAvailable`	boolean	Whether there are more participants to be enumerated.
`cookie`	string (150)	Cookie that must be returned in the next invocation to continue the enumeration.
`participantMediaInfo`	array of structs	Each member of the array is a struct that defines the media token usage for the enumerated participant. The array may be empty if there is no data to return. See Table 102: Participant media information struct members [p.75].

Table 102: Participant media information struct members

Parameter name	Type	Description
`participantID`	string (50)	Participant identifier assigned by the TelePresence Server. See Identifiers and client references [p.11].

Table 102: Participant media information struct members (continued)

Parameter name	Type	Description
conferenceID	string (50)	Conference identifier assigned by the TelePresence Server. See Identifiers and client references [p.11].
participantReference	string (50)	Client reference string. See Identifiers and client references [p.11]. Absent if empty.
mainVideoTokenInfo	struct	A struct that defines the tokens for main video. See Table 103: Participant token information struct members [p.76].
extendedVideoTokenInfo	struct	A struct that defines the tokens for extended video (content). See Table 103: Participant token information struct members [p.76].
audioTokenInfo	struct	A struct that defines the tokens for audio. See Table 103: Participant token information struct members [p.76].
creditsConfigured	integer (>= 0)	Number of credits configured for the participant.
creditsFarEnd	integer (>= 0)	Number of credits required to match far end capability. Absent if the far-end capabilites are not yet known, or if some calls are not yet established.
creditsNearEnd	integer (>= 0)	Number of credits required to match near end capability. Absent if the far-end capabilites are not yet known, or if some calls are not yet established.

Table 103: Participant token information struct members

Parameter name	Type	Description
maxTokensConfigured	integer (>= 0)	Maximum number of tokens with respect to conference or participant configuration.
maxTokensPerChannelConfigured	integer (>= 0)	Maximum number of tokens per channel with respect to the conference or participant configuration. Not returned if **maxTokensPerChannelConfiguredUnlimited** is returned (**true**).
maxTokensPerChannelConfiguredUnlimited	boolean	Whether there is an unlimited maximum number of tokens per channel with respect to the conference or participant configuration. Not returned if **maxTokensPerChannelConfigured** is returned, in which case it is implicitly **false**.
maxTokensPerChannelFarEnd	integer (>= 0)	Maximum number of tokens per channel with respect to the capability of the far end. If the far end has a range of capabilities, these correspond to the maxima. Absent if the far-end capabilites are not yet known, or if some calls are not yet established.

Table 103: Participant token information struct members (continued)

Parameter name	Type	Description
`maxTokensFarEnd`	integer (>= 0)	Maximum number of tokens with respect to the capability of the far end. If the far end has a range of capabilities, these correspond to the maxima. Absent if the far-end capabilites are not yet known, or if some calls are not yet established.
`maxTokensPerChannelNearEnd`	integer (>= 0)	Maximum number of tokens per channel, advertised by the TelePresence Server. Absent if the far-end capabilites are not yet known, or if some calls are not yet established.
`maxTokensNearEnd`	integer (>= 0)	Maximum number of tokens advertised by the TelePresence Server. Absent if the far-end capabilites are not yet known, or if some calls are not yet established.

flex.participant.modify

Modifies the call attributes, media resources, and display name of the specified participant. Only the parameters that you specify are changed.

If you change the call attributes, your changes apply to all calls for this participant. Media resources are distributed as described in Participant media distribution [p.14].

Note: The token requirements for a call cannot be known prior to instantiation of the call, so no checks are made on **`flex.participant.create`** or **`flex.participant.modify`** to determine if the call will have adequate resources. The client is therefore responsible for ensuring that the call has adequate resources.

Table 104: **`flex.participant.modify`** inputs

Parameter name	Type	Description
`participantID`	string (50)	**Required**. Participant identifier assigned by the TelePresence Server. See Identifiers and client references [p.11].
`displayName`	string (80)	Configured display name for the endpoint. This overrides the endpoint display name setting.
`callAttributes`	struct	This Call attributes struct [p.29] modifies the participant's call attributes. These settings override the conference default call attribute settings. See How call attributes are derived [p.29].
`participantMediaResources`	struct	This Participant media resources struct [p.28] modifies the participant's media resource configuration. These settings override the conference default participant media resource configuration. If present, this struct updates all participant media configuration settings: unspecified optional fields are set to their default values.

flex.participant.query

Returns the parameters of the specified participant.

Table 105: **flex.participant.query** inputs

Parameter name	Type	Description
participantID	string (50)	Participant identifier assigned by the TelePresence Server. See Identifiers and client references [p.11].

Table 106: **flex.participant.query** returned data

Parameter name	Type	Description
participantID	string (50)	Participant identifier assigned by the TelePresence Server. See Identifiers and client references [p.11].
conferenceID	string (50)	Conference identifier. See Identifiers and client references [p.11].
PIN	string (40)	Numeric PIN.
callAttributes	struct	This is the previously specified or inherited Call attributes struct [p.29] of the queried participant. See How call attributes are derived [p.29].
participantMediaResources	struct	This is the previously specified or inherited Participant media resources struct [p.28] of the queried participant..
calls	array of structs	Each member of the array is a Participant call definition struct [p.35] that contains the specifications of one of this participant's calls. The array will have at least one member and may have up to four.
camerasCrossed	boolean	Whether cameras in the group of endpoints specified by **calls** are crossed.
audioIndex	integer (>= 0)	Position in the **calls** array of the call that will receive audio.
contentIndex	integer (>= 0)	Position in the **calls** array of the call that will receive content.
cascadeRole	string	One of **cascadeNone**, **cascadeMaster**, or **cascadeSlave**. See Table 19: Cascade roles enumerated type [p.25].
participantReference	string (50)	Client reference string. See Identifiers and client references [p.11]. Returned if string length > 0.
displayName	string (80)	Configured display name for the endpoint. This overrides the endpoint display name setting. Returned if string length > 0.
dtmf	string (127)	Valid DTMF [p.19] characters in a sequence that is sent to the call nominated by the **audioIndex**. This sequence is only sent when dialing out and is not sent for incoming calls.
callerName	string (80)	Calling name seen by the endpoint. Not used for incoming calls. Returned if string length > 0.
callerAddress	string (80)	Calling address seen by the endpoint. Not used for incoming calls. Returned if string length > 0.

flex.participant.requestDiagnostics

Request call diagnostics for participants. If the participant has no active calls, the TelePresence Server returns fault code 56. See Fault codes [p.100].

Table 107: flex.participant.requestDiagnostics inputs

Parameter name	Type	Description
`participantID`	string (50)	**Required**. Participant identifier assigned by the TelePresence Server. See Identifiers and client references.
`receiverURI`	string (255)	**Required**. Fully-qualified `http` or `https` URI (for example, http://example.com:5050/RPC2 or https://example.com:5050/RPC2) to which the diagnostics are sent. If no port number is specified, the device uses the protocol defaults (80 and 443 respectively).
`sourceIdentifier`	string (255 ASCII)	Source identifier. If supplied, the identifier will be returned along with the participant diagnostics. If absent, the unit's Port A MAC address is given.

Asynchronous reply

`flex.participant.requestDiagnostics` works asynchronously because the required information is not available immediately. Therefore, when the query is made for a particular participant (identified by `participantID`), a `receiverURI` needs to be provided. The diagnostics are sent back to the `receiverURI`. The message sent back is an XML-RPC methodCall with methodName `participantDiagnosticsResponse` and contains `audioRx`, `audioTx`, `auxiliaryAudioRx`, `auxiliaryAudioTx`, `videoRx`, `videoTx`, `contentVideoRx`, and `contentVideoTx` arrays, each of which contain a number of structs (one for each stream present). The member parameters of each struct type are described below.

The TelePresence Server can handle up to 10 concurrent asynchronous requests of this type, so this command may fail with fault code 203 if the number of pending requests exceeds this limit.

Table 108: `flex.participant.requestDiagnostics` asynchronously returned data

Parameter name	Type	Description
`participantID`	string (50)	Identifier of the participant to which these diagnostics relate. See Identifiers and client references.
`sourceIdentifier`	string	Source identifier provided in the original request. If absent, the unit's Ethernet A MAC address is given.
`audioRx`	array	Array of `audioRx` stream structs. See Table 109: audioRx stream struct members [p.80].
`audioTx`	array	Array of `audioTx` stream structs. See Table 110: audioTx stream struct members [p.80].
`auxiliaryAudioRx`	array	Array of `auxiliaryAudioRx` stream structs. See Table 111: auxiliaryAudioRx stream struct members [p.81].
`auxiliaryAudioTx`	array	Array of `auxiliaryAudioTx` stream structs. See Table 112: auxiliaryAudioTx stream struct members [p.81].

Table 108: flex.participant.requestDiagnostics asynchronously returned data (continued)

Parameter name	Type	Description
videoRx	array	Array of **videoRx** stream structs. See Table 113: videoRx stream struct members [p.82].
videoTx	array	Array of **videoTx** stream structs. See Table 114: videoTx stream struct members [p.82].
contentVideoRx	array	Array of **contentRx** stream structs. See Table 115: contentVideoRx stream struct members [p.83].
contentVideoTx	array	Array of **contentTx** stream structs. See Table 116: contentVideoTx stream struct members [p.84].

Table 109: **audioRx** stream struct members

Parameter name	Type	Description
codec	string	Codec in use.
encrypted	boolean	Whether the stream data is encrypted.
channelBitRate	integer	Bit rate of the channel in bits per second (bps).
jitter	integer	Current jitter in this stream, measured in milliseconds (ms).
energy	integer	Level of the signal, measured in decibels (dB).
packetsReceived	integer	Count of packets received in this stream.
packetErrors	integer	Count of packets with errors in this stream.
packetsMissing	integer	Count of packets missing from this stream.
framesReceived	integer	Count of frames received in this stream.
frameErrors	integer	Count of frames with errors in this stream.
muted	boolean	Whether the stream is muted.
clearPathOverhead	integer	Only returned if ClearPath has been negotiated. The percentage of FEC overhead in this media stream. The value 50, for example, means that one FEC packet is used to protect every two media packets.
clearPathRecovered	integer	Only returned if ClearPath has been negotiated. The number of media packets recovered using FEC.

Table 110: **audioTx** stream struct members

Parameter name	Type	Description
codec	string	Codec in use.
encrypted	boolean	Whether the stream data is encrypted.
channelBitRate	integer	Bit rate of the channel in bits per second (bps).
packetsSent	integer	Count of packets sent in this stream.

Table 110: audioTx stream struct members (continued)

Parameter name	Type	Description
`muted`	boolean	Whether the stream is muted.
`packetsLost`	integer	The number of packets lost from this stream, as reported by RTCP from the far end.
`clearPathOverhead`	integer	Only returned if ClearPath has been negotiated. The percentage of FEC overhead in this media stream. The value 50, for example, means that one FEC packet is used to protect every two media packets.
`clearPathRecovered`	integer	Only returned if ClearPath has been negotiated. The number of media packets recovered using FEC, as reported by RTCP from the far end.

Table 111: **`auxiliaryAudioRx`** stream struct members

Parameter name	Type	Description
`codec`	string	Codec in use.
`encrypted`	boolean	Whether the stream data is encrypted.
`channelBitRate`	integer	Bit rate of the channel in bits per second (bps).
`jitter`	integer	Current jitter in this stream, measured in milliseconds (ms).
`energy`	integer	Level of the signal, measured in decibels (dB).
`packetsReceived`	integer	Count of packets received in this stream.
`packetErrors`	integer	Count of packets with errors in this stream.
`packetsMissing`	integer	Count of packets missing from this stream.
`framesReceived`	integer	Count of frames received in this stream.
`frameErrors`	integer	Count of frames with errors in this stream.
`muted`	boolean	Whether the stream is muted.

Table 112: **`auxiliaryAudioTx`** stream struct members

Parameter name	Type	Description
`codec`	string	Codec in use.
`encrypted`	boolean	Whether the stream data is encrypted.
`channelBitRate`	integer	Bit rate of the channel in bits per second (bps).
`packetsSent`	integer	Count of packets sent in this stream.
`muted`	boolean	Whether the stream is muted.

Table 113: **`videoRx`** stream struct members

Parameter name	Type	Description
`codec`	string	Codec in use.
`height`	integer	Height of the stream, in pixels.
`width`	integer	Width of the stream, in pixels.
`encrypted`	boolean	Whether the stream data is encrypted.
`channelBitRate`	integer	Bit rate of the channel in bits per second (bps).
`expectedBitRate`	integer	Expected bit rate of this stream, in bits per second (bps).
`expectedBitRateReason`	string	One of: **`viewedSize`**, **`errorPackets`**, or **`notLimited`**.
`actualBitRate`	integer	Measured bit rate of this stream, in bits per second (bps).
`jitter`	integer	Current jitter in this stream, measured in milliseconds (ms).
`packetsReceived`	integer	Count of packets received in this stream.
`packetErrors`	integer	Count of packets with errors in this stream.
`framesReceived`	integer	Count of frames received in this stream.
`frameErrors`	integer	Count of frames with errors in this stream.
`frameRate`	integer	Number of frames being received per second.
`fastUpdateRequestsSent`	integer	Number of fast update requests sent.
`muted`	boolean	Whether the stream is muted.
`clearPathOverhead`	integer	Only returned if ClearPath has been negotiated. The percentage of FEC overhead in this media stream. The value 50, for example, means that one FEC packet is used to protect every two media packets.
`clearPathRecovered`	integer	Only returned if ClearPath has been negotiated. The number of media packets recovered using FEC.
`clearPathLTRFRepaired`	integer	Only returned if ClearPath has been negotiated. The number of frames repaired by referencing the long-term reference frames embedded in this stream.

Table 114: **`videoTx`** stream struct members

Parameter name	Type	Description
`codec`	string	Codec in use.
`height`	integer	Height of the stream, in pixels.
`width`	integer	Width of the stream, in pixels.
`encrypted`	boolean	Whether the stream data is encrypted.
`channelBitRate`	integer	Bit rate of the channel in bits per second (bps).

Table 114: videoTx stream struct members (continued)

Parameter name	Type	Description
configuredBitRate	integer	Configured bit rate of the channel (in bps), see **configuredBitRateReason** for why this differs from **channelBitRate**.
configuredBitRateReason	string	One of: **aggregateBandwidth**, **flowControl**, or **notLimited**.
actualBitRate	integer	Measured bit rate of this stream, in bits per second (bps).
packetsSent	integer	Count of packets sent in this stream.
frameRate	integer	Number of frames being sent per second.
fastUpdateRequestsReceived	integer	Number of fast update requests received.
muted	boolean	Whether the stream is muted.
packetsLost	integer	The number of packets lost from this stream, as reported by RTCP from the far end.
clearPathOverhead	integer	Only returned if ClearPath has been negotiated. The percentage of FEC overhead in this media stream. The value 50, for example, means that one FEC packet is used to protect every two media packets.
clearPathRecovered	integer	Only returned if ClearPath has been negotiated. The number of media packets recovered using FEC, as reported by RTCP from the far end.
clearPathLTRF	boolean	Only returned if ClearPath has been negotiated. **true** if long-term reference frames are being inserted in this stream.

Table 115: **contentVideoRx** stream struct members

Parameter name	Type	Description
codec	string	Codec in use.
height	integer	Height of the stream, in pixels.
width	integer	Width of the stream, in pixels.
encrypted	boolean	Whether the stream data is encrypted.
channelBitRate	integer	Bit rate of the channel in bits per second (bps).
expectedBitRate	integer	Expected bit rate of this stream, in bits per second (bps).
expectedBitRateReason	string	One of: **viewedSize**, **errorPackets**, or **notLimited**.
actualBitRate	integer	Measured bit rate of this stream, in bits per second (bps).
jitter	integer	Current jitter in this stream, measured in milliseconds (ms).
packetsReceived	integer	Count of packets received in this stream.
packetErrors	integer	Count of packets with errors in this stream.

Table 115: contentVideoRx stream struct members (continued)

Parameter name	Type	Description
framesReceived	integer	Count of frames received in this stream.
frameErrors	integer	Count of frames with errors in this stream.
frameRate	integer	Number of frames being received per second.
fastUpdateRequestsSent	integer	Number of fast update requests sent.
clearPathOverhead	integer	Only returned if ClearPath has been negotiated. The percentage of FEC overhead in this media stream. The value 50, for example, means that one FEC packet is used to protect every two media packets.
clearPathRecovered	integer	Only returned if ClearPath has been negotiated. The number of media packets recovered using FEC.
clearPathLTRFRepaired	integer	Only returned if ClearPath has been negotiated. The number of frames repaired by referencing the long-term reference frames embedded in this stream.

Table 116: **contentVideoTx** stream struct members

Parameter name	Type	Description
codec	string	Codec in use.
height	integer	Height of the stream, in pixels.
width	integer	Width of the stream, in pixels.
encrypted	boolean	Whether the stream data is encrypted.
channelBitRate	integer	Bit rate of the channel in bits per second (bps).
configuredBitRate	integer	Configured bit rate of the channel (in bps), see **configuredBitRateReason** for why this differs from **channelBitRate**.
configuredBitRateReason	string	One of: **aggregateBandwidth**, **flowControl**, or **notLimited**.
actualBitRate	integer	Measured bit rate of this stream, in bits per second (bps).
packetsSent	integer	Count of packets sent in this stream.
frameRate	integer	Number of frames being sent per second.
fastUpdateRequestsReceived	integer	Number of fast update requests received.
packetsLost	integer	The number of packets lost from this stream, as reported by RTCP from the far end.
clearPathOverhead	integer	Only returned if ClearPath has been negotiated. The percentage of FEC overhead in this media stream. The value 50, for example, means that one FEC packet is used to protect every two media packets.

Table 116: contentVideoTx stream struct members (continued)

Parameter name	Type	Description
`clearPathRecovered`	integer	Only returned if ClearPath has been negotiated. The number of media packets recovered using FEC, as reported by RTCP from the far end.
`clearPathLTRF`	boolean	Only returned if ClearPath has been negotiated. **`true`** if long-term reference frames are being inserted in this stream.

flex.participant.requestPreview

Requests JPEG previews of video streams to or from the specified participant.

`flex.participant.requestPreview` works asynchronously because participant previews are not available immediately. Therefore when the request is made for a particular participant (identified by **`participantID`**), a **`receiverURI`** needs to be provided.

The TelePresence Server can handle up to 10 concurrent asynchronous requests of this type, so this command may fail with fault code 203 if the number of pending requests exceeds this limit.

Table 117: **`flex.participant.requestPreview`** inputs

Parameter name	Type	Description
`participantID`	string (50)	**Required**. Participant identifier assigned by the TelePresence Server. See Identifiers and client references [p.11].
`receiverURI`	string (255)	**Required**. Fully-qualified **`http`** or **`https`** URI (for example, http://example.com:5050/RPC2 or https://example.com:5050/RPC2) to which the previews are sent. If no port number is specified, the device uses the protocol defaults (80 and 443 respectively).
`streams`	array of structs	**Required**. Each member of the array is a Table 118: Stream struct inputs [p.85] that identifies which stream to preview. You must specify at least one stream and may specify up to four (if it is a grouped endpoint).
`sourceIdentifier`	string (255) ASCII characters only	Source identifier. If supplied, the identifier will be returned along with the previews. If absent, the MAC address of Port A is used.

Table 118: Stream struct inputs

Parameter name	Type	Description
`streamIdentifier`	string	**Required**. Video stream to preview. One of **`rxMainVideo`**, **`txMainVideo`**, or **`extendedVideo`**. In the case of **`extendedVideo`**, the choice of **`incoming`** or **`outgoing`** is decided by the TelePresence Server depending on what is currently active, and this is returned in the response.

Table 118: Stream struct inputs (continued)

Parameter name	Type	Description
`position`	integer (>= 0 and <= {number of screens} - 1)	This parameter must always be supplied unless the **`streamIdentifier`** is **`extendedVideo`**. The parameter is only ever valid between 0 and 3, and defines the position of the endpoint stream within the group/multiscreen endpoint. This should always be 0 for a single-screen endpoint. For grouped or multiscreen endpoints, 0 defines the leftmost screen and the number increments from left to right. For example, the right screen of a three-screen endpoint has **`position= 2`**.
`maxWidth`	integer (>= 88)	Maximum width of generated preview. Useful range 88-176 (pixels). Setting **`maxWidth`** > 176 will not return a wider image. Default: 88.
`maxHeight`	integer (>= 72)	Maximum height of generated preview. Useful range 72-144 (pixels). Setting **`maxHeight`** > 144 will not return a taller image. Default: 72.

Examples of circumstances that cause this command to fail include the following:

- **`streamIdentifier`** is invalid (invalid parameter).
- **`position`** does not exist for the participant (invalid parameter).
- Values of **`maxWidth`** and **`maxHeight`** are invalid (invalid parameter).
- There are too many outstanding requests for previews (<Fault 203: 'too many asynchronous requests'>).
- **`participantID`** is invalid (no such participant).
- **`receiverURI`** is not a valid URI (invalid parameter).
- **`streams`** arrays is empty (invalid parameter).
- There is no active call in the slot indicated by **`position`**(<Fault 56: 'absent participant active call'>).

The maximum number of streams that are available to be requested is:

- 1 + (2 * maximum_number_of_calls_per_participant)
  Where
  - 1 stream is for extended video
  - 2 streams per screen, incoming and outgoing.

For example, if the **`maxCallsPerParticipant`** returned by flex.resource.query is 4, a maximum of 9 streams are available to be requested.

Asynchronous reply

If the request is successful, the previews of the requested streams are sent back to the **`receiverURI`**. The message sent back is an XML-RPC methodCall with methodName **`participantPreviewResponse`** and contains an array of **`preview`** structs - one for each **`stream`** supplied in the initial request.

Table 119: **`participantPreviewResponse`** data

Parameter name	Type	Description
`participantID`	string (50)	Participant identifier assigned by the TelePresence Server. See Identifiers and client references [p.11].
`sourceIdentifier`	string (255)	Source identifier provided in the original request (or Port A MAC address if not supplied in request).
`streams`	array of structs	Each member of the array is a Table 120: Preview struct members [p.87] which will contain the base64-encoded JPEG preview if it was retrieved.

Table 120: Preview struct members

Parameter name	Type	Description
`status`	string	Either **`ok`**, or one of the following strings giving the reason for failing to obtain a preview of the stream: ■ **`audioOnly`**: endpoint is audio-only and is not capable of receiving video. ■ **`noCurrentPresentation`**: currently no active extended video channel (conference has no active presentation stream). ■ **`contentInMain`**: there is active extended video, but this endpoint is not capable of receiving it - presentation is currently being displayed in rxMainVideo stream. ■ **`internalError`**: unexpected error when trying to generate preview. The two content-related statuses are returned only if the requested stream is an extended video stream.
`direction`	string	Whether the preview is of an **`rx`** (incoming) or **`tx`** (outgoing) stream.
`context`	string	Whether preview is of the **`main`** or **`extended`** stream.
`position`	integer	Position of stream starting from 0, going from left to right. For example, the right screen of a T3 would be **`position`** = 2.
`preview`	base64	Base64 encoded JPEG binary data (only valid if status is **`ok`**). The image is constrained by **`maxWidth`** and **`maxHeight`**, and will be the size requested (up to 176x144), although the preview may not fill the returned image.

flex.participant.sendDTMF

Sends the specified DTMF sequence to the endpoint nominated as the audio transmitter and receiver.

Table 121: flex.participant.sendDTMF inputs

Parameter name	Type	Description
`participantID`	string (50)	**Required**. Participant identifier assigned by the TelePresence Server. See Identifiers and client references [p.11].
`dtmf`	string (127)	**Required**. Valid DTMF [p.19] characters in a sequence to send to this participant.

This command may fail with <Fault 56: 'absent participant active call'> if the call that has been nominated as the audio transmitter and receiver is absent.

flex.participant.sendUserMessage

Sends the specified message to a particular participant. For multi-call participants, the message is sent to the call in the center.

The following table lists the input parameters that are required for this command.

Table 122: **`flex.participant.sendUserMessage`** inputs

Parameter name	Type	Description
`participantID`	string (50)	**Required**. Participant identifier assigned by the TelePresence Server. See Identifiers and client references [p.11].
`message`	string (500)	**Required**. Message to display.
`position`	integer (1–9)	Position on display: | 1 2 3 | | 4 5 6 | | 7 8 9 | Default: 5.
`duration`	integer (>0)	Duration, in seconds, for the message to display on the endpoint. The TelePresence Server will accept **`0`** but the behavior is undefined in this case. Default: 30.

flex.participant.setImportant

Designates the specified participant as the important participant. This may result in importance being taken away from another participant in the same conference, even if the participant has no active calls.

Table 123: **`flex.participant.setImportant`** inputs

Parameter name	Type	Description
`participantID`	string (50)	**Required**. Participant identifier assigned by the TelePresence Server. See Identifiers and client references [p.11].

flex.participant.setMute

Changes the muting states of the specified participant for incoming and outgoing audio and video streams. The muting state is only changed for fields that are specified in the command.

Table 124: **`flex.participant.setMute`** inputs

Parameter name	Type	Description
`participantID`	string (50)	**Required**. Participant identifier assigned by the TelePresence Server. See Identifiers and client references [p.11].
`audioRxMute`	boolean	Whether audio from the endpoint is muted.
`videoRxMute`	boolean	Whether the main video from the endpoint is muted.
`audioTxMute`	boolean	Whether audio to the endpoint is muted.
`videoTxMute`	boolean	Whether the main video to the endpoint is muted.

flex.participant.status

Returns the status of the specified participant.

Table 125: **`flex.participant.status`** inputs

Parameter name	Type	Description
`participantID`	string (50)	**Required**. Participant identifier assigned by the TelePresence Server. See Identifiers and client references [p.11].

Table 126: **`flex.participant.status`** returned data

Parameter name	Type	Description
`participantID`	string (50)	Participant identifier assigned by the TelePresence Server. See Identifiers and client references [p.11].
`participantReference`	string (50)	Client reference string. See Identifiers and client references [p.11]. Returned if string length > 0.
`displayName`	string (80)	Participant display name. Returned if string length > 0.
`conferenceID`	string (50)	Identifier of the conference to which the participant is connected or is in the process of connecting. See Identifiers and client references [p.11].
`accessLevel`	string	Access level assigned to the participant. See Table 7: Access level enumerated type [p.22].
`participantMediaInfo`	struct	The Table 127: Participant media information struct members [p.90] details the media resources and credits allocated to this participant.
`important`	boolean	Whether the participant is important.

Table 126: flex.participant.status returned data (continued)

Parameter name	Type	Description
calls	array of structs	Each member of the array is a struct that describes the status of one of the participant's calls. The array contains between one and four calls as previously configured for the participant. See Table 128: Participant call status struct members [p.91]. A member struct will be returned empty if the call is configured but does not currently exist. This ensures that the index position of each call is maintained, to assist you in determining the status of individual calls to grouped endpoints.
audioRxMute	boolean	Whether audio from endpoint is muted.
videoRxMute	boolean	Whether main video from endpoint is muted.
audioTxMute	boolean	Whether audio to endpoint is muted.
videoTxMute	boolean	Whether main video to endpoint is muted.
displayName	string (80)	Participant display name. Returned if string length > 0.
layout	string	The display layout that is currently shown on the participant's endpoint. One of **layoutSingle**, **layoutActivepresence**, **layoutProminent**, or **layoutEqual**. Not returned if there are no calls connected for this participant or if no video is being transmitted to the participant (eg. if transmit video is muted). See Table 10: Single screen layout enumerated type [p.23] or Table 11: Multi-screen layout enumerated type [p.23], depending on the type of endpoint.

Table 127: Participant media information struct members

Parameter name	Type	Description
mainVideoTokenInfo	struct	A Participant token information struct for the participant's main video. See Table 127: Participant media information struct members [p.90].
extendedVideoTokenInfo	struct	A Participant token information struct for the participant's extended video. See Table 127: Participant media information struct members [p.90].
audioTokenInfo	struct	A Participant token information struct for the participant's audio. See Table 127: Participant media information struct members [p.90].
creditsConfigured	integer (>= 0)	Number of credits configured for the participant.
creditsFarEnd	integer (>= 0)	Number of credits required to match far end capability. Absent if the far-end capabilites are not yet known, or if some calls are not yet established.
creditsNearEnd	integer (>= 0)	Number of credits required to match near end capability. Absent if the far-end capabilites are not yet known, or if some calls are not yet established.

Table 128: Participant call status struct members

Parameter name	Type	Description
callID	string (50)	Call identifier. See Identifiers and client references [p.11].
conferenceID	string (50)	Identifier of the conference to which the call is connected or is in the process of connecting. See Identifiers and client references [p.11].
conferenceState	string	State of call connection to the conference. See Table 16: Call conference state enumerated type [p.24].
callState	string	State of the call. See Table 17: Call state enumerated type [p.24].
incoming	boolean	Direction of the call: **true** indicates incoming; **false** indicates outgoing.
protocol	string	Call control protocol. See Table 12: Call protocol enumerated type [p.23].
address	string (80)	Address of the endpoint. For outgoing calls, this is the destination of the call.
duration	integer (>= 0)	Duration of the call in seconds. Only returned if a call has been established.
rxBandwidth	integer (>= 0)	Receive bandwidth. Only returned if a call has been established.
txBandwidth	integer (>= 0)	Transmit bandwidth. Only returned if a call has been established.
remoteName	string (80)	Endpoint name supplied by the far end. Only returned for strings of length > 0.

flex.resource.query

Retrieves TelePresence Server resource settings/parameters. This command takes no input parameters.

Deriving the required number of media tokens

You can derive the number of media tokens required to support a resolution by multiplying the required video width and height to get the required video area, and searching for the best fit in the **videoMediaTokenLevels** array. The best fit in this case is the lowest value of **maxVideoArea** that is larger than or equal to the required video area.

Table 129: **flex.resource.query** returned data

Parameter name	Type	Description
maxCalls	integer (>= 0)	Maximum number of active calls.
maxCallsPerParticipant	integer (>= 0)	Maximum number of calls supported for any one participant.
maxParticipants	integer (>= 0)	Maximum number of participants.

Table 129: flex.resource.query returned data (continued)

Parameter name	Type	Description
`maxParticipantsPerConference`	integer (>= 0)	Maximum number of participants supported for any one conference.
`maxConferences`	integer (>= 0)	Maximum number of conferences.
`maxMediaTokensPerChannel`	integer (> 0)	Maximum number of tokens that can be assigned to a channel.
`mediaTokensLimit`	integer (>= 0)	Maximum number of media tokens available. This varies with the number of TelePresence Servers within the cluster.
`mediaTokensAvailable`	integer (>= 0)	Number of media tokens currently available. This varies with the number of active TelePresence Servers within the cluster.
`maxMediaCredits`	integer (>= 0)	Maximum number of credits available. This varies with installed screen licenses.
`mediaTokenLevelsMainVideo`	array of structs	Each member of the array is a Table 130: videoMediaTokenLevel struct members [p.92] struct that defines mediaToken levels for main video. The **`videoMediaTokenLevel`** XML-RPC struct is used to describe levels associated with mediaToken values such as the supported resolution.
`mediaTokenLevelsExtendedVideo`	array of structs	Each member of the array is a Table 130: videoMediaTokenLevel struct members [p.92] struct that defines mediaToken levels for extended video.
`mediaTokenLevelsAudio`	array of structs	Each member of the array is an Table 131: audioMediaTokenLevel struct members [p.93] that defines mediaToken levels for audio. The **`audioMediaTokenLevel`** struct is used to describe levels associated with mediaToken values required for audio channels.
`mediaCreditTokenRanges`	array of integers	Each entry is the top end (inclusive) of a media credit token range. The value of the previous entry + 1 is the bottom end of the range except for the very first range, which starts from 0. See Media credits [p.15].
`minCallBandwidth`	integer (> 0)	Lowest value (bits per second) that will be accepted for call bandwidths.
`maxCallBandwidth`	integer (> 0)	Highest value (bits per second) that will be accepted for call bandwidths.

Table 130: **`videoMediaTokenLevel`** struct members

Parameter name	Type	Description
`numMediaTokens`	integer (>= 0)	Number of media tokens required for the given video resolution and macroblocks per second. See Deriving the required number of media tokens.

Table 130: videoMediaTokenLevel struct members (continued)

Parameter name	Type	Description
`maxVideoArea`	integer (> 0)	Maximum resolution supported. Multiply required video width and height to check if a resolution is supported. A resolution is supported if: **`maxVideoArea >= (requiredVideoWidth * requiredVideoHeight`**).
`maxMBps`	integer (>= 0)	Maximum macroblocks per second.

Table 131: **`audioMediaTokenLevel`** struct members

Parameter name	Type	Description
`numMediaTokens`	integer (>= 0)	Number of media tokens per channel.
`stereo`	boolean	Whether the channel is stereo.

flex.resource.status

Returns resource usage information. This command takes no input parameters.

Table 132: flex.resource.status returned data

Parameter name	Type	Description
`numCalls`	integer (>= 0)	Number of active calls.
`numParticipants`	integer (>= 0)	Number of active participants.
`numConferences`	integer (>= 0)	Number of active conferences.
`configuredMediaTokens`	integer (>= 0)	Total number of media tokens configured for use. That is, the sum of the tokens that have been configured for all participants in all conferences.
`allocatedMediaTokens`	integer (>= 0)	Total number of allocated media tokens. That is, the sum of tokens required for all participants in all conferences.
`configuredMediaCredits`	integer (>= 0)	Total number of license credits configured for use. That is, the sum of the credits that have been configured for all participants in all conferences.
`allocatedMediaCredits`	integer (>= 0)	Total number of allocated media credits. That is, the sum of credits required for all participants in all conferences.

system.info

Returns the current status of the queried system. This command takes no input parameters.

Table 133: **`system.info`** returned data

Parameter name	Type	Description
`platform`	string	The TelePresence Server's platform, as it appears in **system.xml**.
`cpuModel`	string	The TelePresence Server's Hypervisor CPU model, as it appears in **system.xml**. (Cisco TelePresence Server on Virtual Machine only)
`cpuCount`	integer	The TelePresence Server's number of Virtual CPUs. (Cisco TelePresence Server on Virtual Machine only)
`cpuAvx`	boolean	The TelePresence Server's support for AVX instruction. (Cisco TelePresence Server on Virtual Machine only)
`operationMode`	string	One of **`standalone`** (locally managed), **`flexible`** (remotely managed), or **`slave`** (slave blade in a cluster).
`licenseMode`	string	Depends on the value of **`operationMode`**: Either **`HD`** or **`fullHD`**, if **`operationMode`** is **`standalone`** Always **`flexible`** if **`operationMode`** is **`flexible`** Absent if **`operationMode`** is **`slave`**
`numControlledServers`	integer	Number of TelePresence Servers controlled by this unit (including itself).
`clusterType`	string	The cluster status of this device. One of **`master`**, **`slave`**, or **`unclustered`**.
`gateKeeperOK`	boolean	Whether the gatekeeper is configured and registered.
`tpsNumberOK`	integer	Number of configured and active TelePresence Servers.
`tpdVersion`	string	TelePresence Server version number.
`tpdName`	string	TelePresence Server system name.
`tpdUptime`	integer	Period of time (in seconds) that has passed since the system booted.
`tpdSerial`	string	TelePresence Server serial number.
`makeCallsOK`	boolean	In flexible (remotely managed) mode, this value is always **`false`** and should be ignored.
`portsVideoTotal`	integer	In flexible (remotely managed) mode, this value is always 0 and should be ignored.
`portsVideoFree`	integer	In flexible (remotely managed) mode, this value is always 0 and should be ignored.
`portsAudioTotal`	integer	In flexible (remotely managed) mode, this value is always 0 and should be ignored.
`portsAudioFree`	integer	In flexible (remotely managed) mode, this value is always 0 and should be ignored.
`portsContentTotal`	integer	In flexible (remotely managed) mode, this value is always 0 and should be ignored.

Table 133: system.info returned data (continued)

Parameter name	Type	Description
`portsContentFree`	integer	In flexible (remotely managed) mode, this value is always 0 and should be ignored.
`maxConferenceSizeVideo`	integer	In flexible (remotely managed) mode, this value is always 0 and should be ignored.
`maxConferenceSizeAudio`	integer	In flexible (remotely managed) mode, this value is always 0 and should be ignored.
`maxConferenceSizeContent`	integer	In flexible (remotely managed) mode, this value is always 0 and should be ignored.
`softwareVersion`	string	Software version string eg. *4.0(1.25)*

Related information

system.xml on 8710 and 7010

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>
    <product>TS</product>
    <platform>8710</platform>
    <productDisplayName>Cisco TelePresence Server</productDisplayName>
    <platformDisplayName>8710</platformDisplayName>
    <serial>SM021037</serial>
    <softwareVersion>3.1(1.45)</softwareVersion>
    <buildVersion>13.1(1.45)</buildVersion>
    <hostName>A host name</hostName>
    <ipAddress>198.51.100.14</ipAddress>
    <ipAddressV6>2001:DB8::81b7</ipAddressV6>
    <macAddress>BA:98:76:54:32:10</macAddress>
    <gatekeeperUsage>Yes</gatekeeperUsage>
    <gatekeeperAddress>mainvcs.test.lal</gatekeeperAddress>
    <gatekeeperIds>dt12b7,dt12b7-l,dt12b7-c,dt12b7-r</gatekeeperIds>
    <sipRegistrarUsage>Yes</sipRegistrarUsage>
    <sipRegistrarAddress>mainvcs.test.lal</sipRegistrarAddress>
    <sipRegistrarDomain>test.lal</sipRegistrarDomain>
    <sipTrunkUsage>No</sipTrunkUsage>
    <sipTrunkAddress/>
    <sipTrunkDomain/>
    <isMaster>Yes</isMaster>
    <clusterType>unclustered</clusterType>
    <totalVideoPorts>12</totalVideoPorts>
    <totalContentPorts>12</totalContentPorts>
    <totalAudioOnlyPorts>10</totalAudioOnlyPorts>
    <uptimeSeconds>230641</uptimeSeconds>
  </system>
```

Table 134: System XML contents

Node name	**Node contents**
manufacturer	TANDBERG
model	Telepresence Server <model number> eg. *Telepresence Server 8710*
product	TS
platform	<platform> eg. *Media 310*, *8710*, or *Virtual Machine with 16 vCPUs*
productDisplayName	*Cisco TelePresence Server*. The display name values are subject to change with new software releases, so your application should not rely on them.

Table 134: System XML contents (continued)

Node name	Node contents
platformDisplayName	<platform> eg. *Media 310*, *8710*, or *Virtual Machine with 16 vCPUs*. The display name values are subject to change with new software releases, so your application should not rely on them.
serial	Unique serial number of the unit
softwareVersion	Software version string eg. *4.0(1.25)*
buildVersion	Build number string eg. *13.2(1.25)*
hostName	Host name of the unit
ipAddress	IPv4 address
ipAddressV6	IPv6 address
macAddress	MAC address
gatekeeperUsage	**`Yes`**: gatekeeper usage is enabled **`No`**: gatekeeper usage is disabled
gatekeeperAddress	The gatekeeper host name or IP address
gatekeeperIds	Comma separated list of registered IDs associated with this TelePresence Server and its slaves (omitted if the system is not a master)
sipRegistrarUsage	**`Yes`**: registrar usage is enabled **`No`**: registrar usage is disabled This value is always **`No`** in remotely managed mode.
sipRegistrarAddress	SIP registrar host name / IP address This node is always empty in remotely managed mode.
sipRegistrarDomain	SIP registrar domain This node is always empty in remotely managed mode.
sipTrunkUsage	**`Yes`**: trunk usage is enabled **`No`**: trunk usage is disabled
sipTrunkAddress	SIP trunk host name / IP address
sipTrunkDomain	SIP trunk domain
isMaster	**`Yes`**: this system is a master, or it is unclustered **`No`**: this system is a slave
clusterType	The role of this system in a cluster. May be **`unclustered`**, **`master`**, or **`slave`**
totalVideoPorts	Total number of video ports This value is always 0 and should be ignored.
totalContentPorts	Total number of video content ports This value is always 0 and should be ignored.
totalAudioOnlyPorts	Total number of audio-only ports This value is always 0 and should be ignored.
uptimeSeconds	System uptime in seconds

system.xml on Media 310/320

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>Cisco</manufacturer>
    <model>TelePresence Server on Media 320</model>
    <product>TS</product>
    <platform>Media 320</platform>
    <productDisplayName>Cisco TelePresence Server</productDisplayName>
    <platformDisplayName>Media 320</platformDisplayName>
    <serial>SUK1702000D</serial>
    <softwareVersion>3.1(1.49)</softwareVersion>
    <buildVersion>13.1(1.49)</buildVersion>
    <hostName>HostName1</hostName>
    <ipAddress>198.51.100.15</ipAddress>
    <ipAddressV6>2001:DB8::81b8</ipAddressV6>
    <macAddress>01:23:45:67:89:AB</macAddress>
    <clusterType>unclustered</clusterType>
  </system>
```

Table 135: System XML contents

Node name	Node contents
manufacturer	Cisco
model	TelePresence Server on <platform> eg. *TelePresence Server on Media 310*
product	TS
platform	<platform> eg. *Media 310*, *8710*, or *Virtual Machine with 16 vCPUs*
productDisplayName	*Cisco TelePresence Server*. The display name values are subject to change with new software releases, so your application should not rely on them.
platformDisplayName	<platform> eg. *Media 310*, *8710*, or *Virtual Machine with 16 vCPUs*. The display name values are subject to change with new software releases, so your application should not rely on them.
serial	Unique serial number of the unit
softwareVersion	Software version string eg. *4.0(1.25)*
buildVersion	Build number string eg. *13.2(1.25)*
hostName	Host name of the unit
ipAddress	IPv4 address
ipAddressV6	IPv6 address
macAddress	MAC address
clusterType	The role of this system in a cluster. May be **`unclustered`**, **`master`**, or **`slave`**

system.xml on Virtual Machine

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>Cisco</manufacturer>
    <model>TelePresence Server on Virtual Machine with 16 vCPUs</model>
    <cpuModel>Intel(R) Xeon(R) CPU E5-4650 0 @ 2.70GHz</cpuModel>
    <cpuCount>30</cpuCount>
    <cpuAvx>1</cpuAvx>
    <product>TS</product>
    <platform>Virtual Machine with 16 vCPUs</platform>
    <productDisplayName>Cisco TelePresence Server</productDisplayName>
    <platformDisplayName>Virtual Machine with 16 vCPUs</platformDisplayName>
    <serial>057ED0A9</serial>
    <softwareVersion>4.0(1.25)</softwareVersion>
    <buildVersion>13.2(1.25)</buildVersion>
    <hostName>HostName1</hostName>
    <ipAddress>198.51.100.15</ipAddress>
    <ipAddressV6>2001:DB8::81b8</ipAddressV6>
    <macAddress>01:23:45:67:89:AB</macAddress>
    <clusterType>unclustered</clusterType>
  </system>
```

Table 136: System XML contents

Node name	Node contents
manufacturer	Cisco
model	TelePresence Server on <platform> eg. *TelePresence Server on Virtual Machine with 16 vCPUs*
cpuModel	TelePresence Server's Hypervisor CPU model (e.g. Intel(R) Xeon(R) CPU E5-4650 0 @ 2.70GHz, etc.) (Cisco TelePresence Server on Virtual Machine only)
cpuCount	TelePresence Server's number of virtual CPUs (Cisco TelePresence Server on Virtual Machine only)
cpuAvx	TelePresence Server's support for AVX instruction. *0* if AVX is not supported or *1* if AVX is supported. (Cisco TelePresence Server on Virtual Machine only)
product	TS
platform	<platform> eg. *Media 310*, *8710*, or *Virtual Machine with 16 vCPUs*
productDisplayName	*Cisco TelePresence Server*. The display name values are subject to change with new software releases, so your application should not rely on them.
platformDisplayName	<platform> eg. *Media 310*, *8710*, or *Virtual Machine with 16 vCPUs*. The display name values are subject to change with new software releases, so your application should not rely on them.
softwareVersion	Software version string eg. *4.0(1.25)*

Table 136: System XML contents (continued)

Node name	Node contents
buildVersion	Build number string eg. *13.2(1.25)*
hostName	Host name of the unit
ipAddress	IPv4 address
ipAddressV6	IPv6 address
macAddress	MAC address
clusterType	Always **`unclustered`**. (Cisco TelePresence Server on Virtual Machine does not support clustering.)

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.

Table 137: Fault codes

Fault Code	Description
1	**`method not supported`**. This method is not supported on this device or is unknown.
4	**`no such conference`**. The conference identification given does not match any conference.
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.
7	**`too many participants`**. There are already too many participants configured and no more can be created.
14	**`authorization failed`**. The requested operation is not permitted because the supplied authentication parameters were not recognized.
15	**`insufficient privileges`**. The specified user id and password combination is not valid for the attempted operation.
17	**`call reservation failure`**. There are insufficient free calls/participants to complete/place the requested calls.
18	**`duplicate URI`**. A URI was given, but this URI 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	**`participant limit lower than active`**. New participant limit is lower than current number of participants.
33	**`out of range`**. A call supplied a value that is outside of the allowed range for this parameter.
34	**`internal error`**. An error occurred while processing the API request.

Table 137: Fault codes (continued)

35	**`string is too long`**. The call supplied a string parameter that was longer than allowed.
50	**`binary data array is too long`**. The call supplied binary data that was longer than allowed.
52	**`no available SIP registration`**. There is no available SIP registration to complete the call.
53	**`insufficient media credits or tokens`**. Fewer media credits or tokens were supplied than were required to complete the call.
55	**`malformed cookie`**. The supplied cookie could not be read.
56	**`no active participant call`**. The participant does not currently have any active calls, or has no active call at the specified position.
57	**`some participants failed`**. The API request could not be completed for some participants in a conference.
58	**`incorrect media credits or tokens`**. Fewer media credits or tokens were supplied than are currently in use.
61	The removal of a feature or license key failed for one of several reasons. The fault code message will vary depending on the underlying cause of the failure.
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 an enumerated type contains an invalid value.
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.
202	**`Product needs its activation feature key`**. This request requires that the product is activated.
203	**`Too many asynchronous requests`**. The TelePresence Server is currently dealing with the maximum number of asynchronous requests of this type. Please retry this request later.
204	**`Too many invalid keys entered. Wait 5 seconds to retry`**. The TelePresence Server will not currently accept more requests to add feature keys.

Example XML-RPC response to `flex.conference.create`

Method call

```
<?xml version='1.0' encoding='UTF-8'?>
<methodCall>
  <methodName>flex.conference.create</methodName>
  <params>
    <param>
      <value>
        <struct>
          <member>
            <name>authenticationPassword</name>
            <value>
              <string></string>
            </value>
          </member>
          <member>
            <name>conferenceName</name>
            <value>
              <string>Flex API conference</string>
            </value>
          </member>
          <member>
            <name>participantMediaResources</name>
            <value>
              <struct>
                <member>
                  <name>mediaTokensAudio</name>
                  <value>
                    <struct>
                      <member>
                        <name>total</name>
                        <value>
                          <int>96</int>
                        </value>
                      </member>
                    </struct>
                  </value>
                </member>
                <member>
                  <name>mediaTokensExtendedVideo</name>
                  <value>
                    <struct>
                      <member>
                        <name>total</name>
                        <value>
                          <int>1920</int>
                        </value>
                      </member>
                    </struct>
                  </value>
                </member>
                <member>
                  <name>mediaTokensMainVideo</name>
                  <value>
                    <struct>
                      <member>
                        <name>total</name>
                        <value>
                          <int>1920</int>
                        </value>
                      </member>
                    </struct>
                  </value>
                </member>
                <member>
                  <name>numMediaCredits</name>
                  <value>
                    <int>5040</int>
                  </value>
                </member>
```

```
              </struct>
            </value>
          </member>
          <member>
            <name>authenticationUser</name>
            <value>
              <string>admin</string>
            </value>
          </member>
        </struct>
      </value>
    </param>
  </params>
</methodCall>
```

Method response

```
<?xml version="1.0" encoding="UTF-8"?>
<methodResponse>
  <params>
    <param>
      <value>
        <struct>
          <member>
            <name>conferenceID</name>
            <value>
              <string>b9852090-f5b9-11e1-8ac5-000d071080b8</string>
            </value>
          </member>
        </struct>
      </value>
    </param>
  </params>
</methodResponse>
```

Remotely managed API change history

Table 138: API version 4.0(2.8) change summary

XML-RPC Request / Topic	Parameter	Change
callHome.configure [p.37]	**mode**, **automatic**	New command
callHome.query [p.37]	**mode**, **automatic**	New command
system.info [p.158]	**cpuModel** **cpuCount** **cpuAvx**	Added

Table 139: API version 4.0 change summary

XML-RPC Request / Topic	Parameter	Change
device.feature.add [p.40]	**key**	New command
device.feature.remove [p.40]	**key**	New command

Table 139: API version 4.0 change summary (continued)

XML-RPC Request / Topic	Parameter	Change
device.query [p.43]	**mediaResourceRestarts**, **key**, **expiry**	Added
device.query [p.43]	**currentTime**, **restartTime**, **uptime**	Documentation corrected
Fault codes [p.100]	61, 204	New fault codes
system.info [p.93]	**softwareVersion**	Added
Enumerated types [p.22]	▪ Table 19: Cascade roles enumerated type [p.25] ▪ Table 20: Optimization profiles enumerated type [p.25] ▪ Table 21: Switching mode enumerated type [p.25] ▪ Table 22: Participant connection state enumerated type [p.25] ▪ Table 23: Encryption status enumerated type [p.26] ▪ Table 24: Media status enumerated type [p.26]	Added
Call attributes struct [p.29]	**recordingDeviceIndicateOnly**, **displayLayoutSwitchingMode**, **indicateMuting**, **allowStarSixMuting**	Added
Call attributes struct [p.29]	**videoRxFlowControlOnViewedSize**	When **true**, behavior modified. Flow control now only requested when received stream not viewed by other participants.
flex.conference.create [p.49],flex.conference.query [p.61], flex.conference.modify [p.57]	**optimizationProfile**, **useCustomMutedCanUnmuteMessage**, **customMutedCanUnmuteMessage**, **useCustomMutedCannotUnmuteMessage**, **customMutedCannotUnmuteMessage**	Added
flex.conference.create [p.49], flex.conference.query [p.61], flex.conference.modify [p.57]	**voiceSwitchingSensitivity**	Default modified from 50 to 60.
flex.conference.enumerate [p.54]	**conferenceName**, **presenterID**, **importantID**	Added
flex.conference.status [p.65]	**presenterID**	Added
flex.participant.enumerate [p.73]	**displayName**, **connectionState**	Added
flex.participant.advanced.enumerate [p.66]	Added **encryptionStatus**, **layout**, **mediaStatus**, **rxBandwidth**, **txBandwidth**, and **protocol**	New command. Alternative to **flex.participant.enumerate**
flex.participant.deletions.enumerate [p.72]	**conferenceID**	Added

Table 139: API version 4.0 change summary (continued)

XML-RPC Request / Topic	Parameter	Change
flex.participant.create [p.70], flex.participant.query [p.77]	**`cascadeRole`**	Added
flex.participant.status [p.89]	**`layout`**	Added
Feedback events [p.20]	**`flexParticipantAdvancedEnum`**	New feedback event
system.xml on Virtual Machine [p.99]		New reference topic

Table 140: API version 3.1 change summary

XML-RPC Request / Topic	Parameter	Change
system.info [p.93]	**`clusterType`**	Added
device.query [p.43]	**`activatedLicenses`**	Added
Enumerated types [p.22]	Audio gain modes (**`gainModeDisabled`**, **`gainModeAutomatic`**, **`gainModeFixed`**) Control levels (**`controlNone`**, **`controlLocal`**, **`controlConference`**)	Added
Call attributes struct [p.29]	**`audioReceiveGainMode`**, **`deferConnect`**, **`alwaysReconnect`**, **`displayForceDefaultLayout`**, **`iXEnabled`**	Added
Call attributes struct [p.29]	**`audioReceiveGain`**	Modified. Previously, **`audioReceiveGain`** was always applied. In 3.1, **`audioReceiveGain`** is ignored unless the **`audioReceiveGainMode`** is **`gainModeFixed`**.
Call attributes struct [p.29]	**`maxTransmitPacketSize`**	Description modified.
Participant call definition struct [p.35]	Default values	Documented
Fault codes [p.100] reference topic		Documented
flex.conference.create [p.49], flex.conference.modify [p.57], flex.conference.query [p.61]	**`conferenceDescription`**, **`chairControlLevel`**, **`guestControlLevel`**	Added
flex.conference.create [p.49], flex.conference.modify [p.57], flex.conference.query [p.61]	**`welcomeMessageScreen`**	Modified
flex.participant.create [p.70], flex.participant.query [p.77], flex.participant.sendDTMF [p.87]	**`dtmf`**	Modified

Table 140: API version 3.1 change summary (continued)

XML-RPC Request / Topic	Parameter	Change
flex.participant.deletions.enumerate [p.72]	**extended**, **IDs**, **participantID**, **conferenceID**	Added
flex.participant.media.enumerate [p.75]	**conferenceID**	Added
flex.participant.requestDiagnostics [p.79]	**clearPathOverhead**, **clearPathRecovered**, **packetsLost**, **clearPathLTRF**, **clearPathLTRFRepaired**	Added

Part 2: Standalone operation mode

Part 2 of this guide describes the API available in standalone operation mode (locally managed). For information about the API available in flexible operation mode (remotely managed), refer to Part 1: Flexible operation mode [p.3].

Introduction 109
- Standalone mode API change summary 109
- Design considerations 109

XML-RPC implementation 111
- Transport 111
- Encoding 111
- Message flow 112
- Data types and sizes 114
- HTTP keep-alives 114

API overview 116
- Authentication 116
- Feedback receivers 116

API command reference 119
- Deprecations 120
- callHome.configure 120
- callHome.query 121
- cdrlog.enumerate 121
- cdrlog.query 123
- conference.create 123
- conference.delete 125
- conference.enumerate 125
- conference.invite 128
- conference.senddtmf 133
- conference.sendmessage 133
- conference.sendwarning 134
- conference.set 134
- conference.status 136
- conference.uninvite 143
- device.feature.add 144
- device.feature.remove 144
- device.query 144
- device.network.query 146
- device.health.query 148
- device.restartlog.query 148
- device.restart 149
- feedbackReceiver.configure 149
- feedbackReceiver.query 150
- feedbackReceiver.reconfigure 151
- feedbackReceiver.remove 151
- feedbackReceiver.status 151
- participant.diagnostics 152
- participant.enumerate 155
- participant.set 156

participant.tidylayout 157
system.info 158
Related information 160
system.xml file 160
Fault codes 161
Example XML-RPC response to participant.diagnostics 163
Locally managed API change history 170

Introduction

This document accompanies the latest version of the management API for the Cisco TelePresence Server software when running in standalone (locally managed) mode. The following Cisco TelePresence products support this API when they are running TelePresence Server version 4.0(2.8) and later:

- Cisco TelePresence Server MSE 8710
- Cisco TelePresence Server 7010

Standalone mode API change summary

The latest Cisco TelePresence Server API is version 4.0(2.8). The table below contains a summary of the latest changes to the remotely managed mode API. For changes introduced in older versions, see Locally managed API change history [p.170].

Table 141: API version 4.0(2.8) change summary

XML-RPC Request / Topic	Parameter	Change
callHome.configure [p.120]	**`mode`**, **`automatic`**	New command
callHome.query [p.121]	**`mode`**, **`automatic`**	New command

Design considerations

Every API command that your application sends incurs a processing overhead within the device’s own application. The amount of the overhead varies widely with the type of command and the parameters sent. If the device receives a high number of API commands every second, its performance could be seriously impaired (in the same way as if multiple users simultaneously accessed it via the web interface).

Minimizing API overhead

It is essential to design your application architecture and software so that the processing load on the device application is minimized.

To do this we recommend that you do the following:

- Use a single server to run the API application and to send 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. Then use the server to 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 requests.

These measures provide much more control than having the clients send API commands directly, and will prevent the device performance being impaired by unmanageable numbers of API requests.

Unavailable or irrelevant data

The API is designed to minimize impact on the network when responding to requests, and device responses do not routinely include either irrelevant data or empty data structures where the data is unavailable.

It follows that your application should take responsibility for checking whether a response includes the expected data, and should be designed for graceful handling of situations where the device does not respond with the expected data.

XML-RPC implementation

The API is implemented as messages sent using the XML-RPC protocol. This is a simple protocol for remote procedure calling that uses HTTP (or HTTPS) as the transport and XML as the encoding. XML-RPC is designed to be as simple as possible while allowing for complex data structures to be transmitted, processed and returned. It has no platform or software dependence and was chosen in favor of SOAP (Simple Object Access Protocol) because of its simplicity.

The API implements all parameters and returned data as **`<struct>`** elements, each of which is explicitly named. For example, the device.query call returns the current time as a structure member named currentTime rather than as a single <dateTime.iso8601> value:

```
<member>
  <name>
    currentTime
  </name>
  <value>
    <dateTime.iso8601>
      20130218T10:45:00
    </dateTime.iso8601>
  </value>
</member>
```

Refer to the XML-RPC specification for more information.

Transport

The device implements HTTP/1.1 as defined by RFC 2616. 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.

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"?>`**.

Message flow

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

The example command below is: create conference: 'API Conference' with numeric ID: '971771' and PIN: '123'

Example command

```
<?xml version='1.0'?>
<methodCall>
  <methodName>conference.create</methodName>
  <params>
    <param>
      <value>
        <struct>
          <member>
            <name>authenticationUser</name>
            <value>
              <string>admin</string>
            </value>
          </member>
          <member>
            <name>authenticationPassword</name>
            <value>
              <string></string>
            </value>
          </member>
          <member>
            <name>conferenceName</name>
            <value>
              <string>API Conference</string>
            </value>
          </member>
          <member>
            <name>numericID</name>
            <value>
              <string>971771</string>
            </value>
          </member>
          <member>
            <name>PIN</name>
            <value>
              <string>123</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>conferenceID</name>
            <value>
              <int>10000</int>
            </value>
          </member>
          <member>
            <name>conferenceGUID</name>
            <value>
              <string>62f46be0-c6a3-11e1-9800-000d7c10cc70</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>13</int>
          </value>
        </member>
        <member>
          <name>faultString</name>
          <value>
            <string>invalid PIN</string>
          </value>
        </member>
      </struct>
    </value>
  </fault>
</methodResponse>
```

Data types and sizes

Note: The total size of a request or response is 32 kB. If the TelePresence Server needs to truncate a response it will either provide a mechanism for you to retrieve the remaining data or return an appropriate fault code.

The Cisco TelePresence Server API accepts the following XML-RPC types. The table includes the default sizes that your application can assume unless a more specific limit is given in a parameter description.

Table 142: API data types and sizes

Type	Default size accepted
<string>	31 characters
<int>	Four byte signed (-2147483648 to 2147483647)
<boolean>	`1` or `0`, `true` or `false`
<base64>	Not explicitly limited unless otherwise stated
<dateTime.iso8601>	ISO 8601 format eg. `20140107T13:31:26`
<array>	N/A
<struct>	N/A

HTTP keep-alives

Your application can 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.

API overview

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 the following authentication parameters:

Table 143: Authentication parameters

Parameter name	Type	Description
`authenticationUser`	string	**Required**. User name.
`authenticationPassword`	string	**Required**. User password.

If the user name and password are not recognized by the TelePresence Server, the method call fails with authentication errors.

Feedback receivers

The API allows you to register your application as a feedback receiver. This means that the application does not have to constantly poll the device if it wants to monitor activity. By using feedback events, you can avoid imposing the high loads that polling can cause especially when there are multiple API users.

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.

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

Note: The TelePresence Server expects your application to provide at least an `HTTP 200 OK` status header. The TelePresence Server logs a warning event if it cannot be sure your application received the feedback message.

- Use feedbackReceiver.configure [p.149] to register a receiver to listen for one or more Feedback events [p.117].
- Use feedbackReceiver.query [p.150] to return a list of receivers that are configured on the device.
- Use feedbackReceiver.reconfigure [p.151] to change the configuration of an existing feedback receiver.
- Use feedbackReceiver.remove [p.151] to remove an existing feedback receiver.
- Use feedbackReceiver.status [p.151] to display the status of a specific feedback receiver, and all the events to which it is subscribed.

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 **`feedbackReceiver.reconfigure`**. If it has not been set it 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:

Table 144: Feedback events

Event	Description
cdrAdded	One or more new Call Detail Records have been logged
conferenceActive	One or more conferences have become active (first participant joined)
conferenceFinished	One or more conferences have been deleted
conferenceStarted	One or more conferences have been created
conferenceInactive	One or more conferences have become inactive (last participant left)
configureAck	The source publishes this event to acknowledge that an application has successfully added, reconfigured, or removed a feedback receiver

Table 144: Feedback events (continued)

Event	Description
deviceStatusChanged	Generated when the TelePresence Server is shut down or a feature key is added or removed. Invoke **`device.query`** for more details.
participantConnected	One or more participants have connected to the TelePresence Server
participantDisconnected	One or more participants disconnected from the TelePresence Server
participantJoined	One or more participants have joined a conference
participantLeft	One or more participants have left a conference
receiverDeleted	The feedback receiver receiving this event has been stopped and its configuration deleted or the URI of the feedback receiver has been changed, in which case this event is sent to the previous URI.
receiverModified	The feedback receiver receiving this event has been modified.
restart	The TelePresence Server has restarted or booted.

API command reference

This section contains a reference to each of the commands available when the operation mode is set to **`standalone`**.

The commands are grouped alphabetically by the objects that they query or modify. The following information is provided for each command:

- Description of the command's effect
- Accepted parameters, and whether they are required
- Returned parameters, and whether they are conditionally returned

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

- cdrlog.enumerate [p.121]
- cdrlog.query [p.123]
- conference.create [p.123]
- conference.delete [p.125]
- conference.enumerate [p.125]
- conference.invite [p.128]
- conference.senddtmf [p.133]
- conference.sendmessage [p.133]
- conference.sendwarning [p.134]
- conference.set [p.134]
- conference.status [p.136]
- conference.uninvite [p.143]
- device.health.query [p.148]
- device.network.query [p.146]
- device.query [p.144]
- device.restartlog.query [p.148]
- device.restart [p.149]
- feedbackReceiver.configure [p.149]
- feedbackReceiver.query [p.150]
- feedbackReceiver.reconfigure [p.151]
- feedbackReceiver.remove [p.151]
- feedbackReceiver.status [p.151]
- participant.diagnostics [p.152]
- participant.enumerate [p.155]
- participant.set [p.156]
- participant.tidylayout [p.157]
- system.info [p.158]

Deprecations

The following parameters were deprecated in version 2.2 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.

Table 145: Locally managed mode deprecated parameters

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

callHome.configure

Configures the TelePresence Server to automatically report diagnostic data to Cisco's Call Home service. This feature is disabled by default, but we strongly recommend that you enable it to ensure the best possible support for your device.

Note: The TelePresence Server currently only supports anonymous reporting.

Table 146: **`callHome.configure`** inputs

Parameter name	Type	Description
`mode`	string	Set the Call Home mode. One of **`disabled`** or **`anonymous`**. Can only be set to **`anonymous`** if the encryption feature key is present. Defaults to **`disabled`** if it has never been configured. Omit the parameter to leave the current setting unchanged.
`automatic`	boolean	Controls automatic Call Home. **`true`** enables automatic Call Home. **`false`** disables automatic Call Home. Only has effect when mode is **`anonymous`**. Omit the parameter to leave the current setting unchanged.

callHome.query

Queries the TelePresence Server to retrieve its Call Home configuration. This feature reports diagnostic data to Cisco's Call Home service.

Note: The TelePresence Server currently only supports anonymous reporting.

Table 147: **`callHome.query`** returned data

Parameter name	Type	Description
`mode`	string	Call Home mode. One of **`disabled`** or **`anonymous`**. Defaults to **`disabled`** if it has never been configured.
`automatic`	boolean	**`true`** if automatic Call Home is enabled. **`false`** if automatic Call Home is disabled. Only has effect if mode is **`anonymous`**. Defaults to **`false`** if it has never been configured.

cdrlog.enumerate

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.

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.

Table 148: **`cdrlog.enumerate`** optional or conditional inputs

Parameter name	Type	Description
`index`	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, the device will enumerate events from the beginning of the CDR log.

Table 148: cdrlog.enumerate optional or conditional inputs (continued)

Parameter name	Type	Description
numEvents	integer	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. Fewer events are returned if they are too large to fit into a single response. Clients should look at the **eventsRemaining** parameter in the response and re-enumerate, starting from **nextIndex**, if necessary.
filter	array	An array of strings, which contain the names of event types by which to filter the response. Omit **filter** to return all event types or include a subset of the following: **conferenceStarted**, **conferenceFinished**, **conferenceActive**, **conferenceInactive**, **participantConnected**, **participantJoined**, **participantMediaSummary**, **participantLeft**, **participantDisconnected**.

Table 149: **cdrlog.enumerate** returned data

Parameter name	Type	Description
startIndex	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	integer	Revision number of the data being provided, reusable in a subsequent call to the API.
eventsRemaining	boolean	If **true**,there is more data in the requested enumeration than has been returned in this response.
currentTime	dateTime.iso8601	The system's current time (UTC).
events	array of structs	Each member of the array is a struct that represents a recorded event. The structures all have some common fields (**time**, **type**, **index**) and may have other fields that are specific to the event type.

Events array

The following parameters are common to all CDR log events, but each struct will also contain parameters specific to the event type. See *Cisco TelePresence Conferencing Call Detail Records File Format Reference Guide* for details of all the TelePresence Server's event types.

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

Table 150: Common CDR log event parameters

Parameter name	Type	Description
time	dateTime.iso8601	Date and time when the event was logged; for example, **20110119T13:52:42**.
type	string	Name of the event type.
index	integer	Index of the CDR log message.

Note: The *Cisco TelePresence Conferencing Call Detail Records File Format 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.

cdrlog.query

Returns information about the CDR log. This command takes no input parameters.

Table 151: `cdrlog.query` returned data

Parameter name	Type	Description
`firstIndex`	integer	Index of the oldest stored event.
`numEvents`	integer	Total number of events stored.

conference.create

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

Table 152: `conference.create` inputs

Parameter name	Type	Description
`conferenceName`	string (80)	**Required**. The name that refers to the conference that is the subject of your call or the response from the TelePresence Server.
`persistent`	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. Use `persistent` instead of `permanent` to define conference persistence, even though the TelePresence Server will accept either. ▪ 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`	boolean	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). **Deprecated**. Use `persistent` instead.
`locked`	boolean	Defines whether the conference is locked. Endpoints can not join a locked conference but the conference can invite them in. ▪ true: The conference is locked. ▪ false: The conference is not locked.

Table 152: conference.create inputs (continued)

Parameter name	Type	Description
lockDuration	integer	The period of time (in seconds) from now until the conference lock expires. Requires that **locked** is **true** and ignored otherwise.
numericID	string (80)	Used for registration with H.323 gatekeeper / SIP registrar, and to dial in to the conference.
registerWithGatekeeper	boolean	Defines whether or not this item registers its **numericID** with the H.323 gatekeeper.
registerWithSIPRegistrar	boolean	Defines whether or not this item registers its **numericID** with the SIP registrar.
tsURI	string (80)	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	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	boolean	Defines whether the conference shows the lobby screen.
lobbyMessage	string (500)	The lobby screen message.
useWarning	boolean	Defines whether the conference sends 'This conference is about to end' warning.
audioPortLimit	integer	The limit on the number of audio ports this conference may allow.
videoPortLimit	integer	The limit on the number of video ports this conference may allow.
duration	integer	Period of time (in seconds) until the conference ends and is deleted. This parameter is not allowed if **persistent** is **true**.
automaticGainControl	boolean	Defines whether automatic gain control is enabled. If not specified, the conference default is used.
encryptionRequired	boolean	Defines whether encryption is required for this conference. ▪ true: Encryption is required. ▪ false: Encryption is optional (default).

Table 152: conference.create inputs (continued)

Parameter name	Type	Description
`pin`	string (40)	The PIN for this conference. If associated with a conference, it is a string of numeric digits that must be entered to gain access to that conference. **Note:** A PIN is only valid for incoming calls - no outgoing calls will ever need to enter it. As a result of this, a conference PIN can only be set when the conference has a numeric ID. Trying to set a PIN without a numeric ID will return a fault, and clearing a conference's numeric ID will also clear that conference's PIN.

Table 153: **`conference.create`** returned data

Parameter name	Type	Description
`conferenceGUID`	string	Globally unique identifier of the conference.
`conferenceID`	integer	Unique conference identifier. **Deprecated**. Use **`conferenceGUID`** instead.

conference.delete

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

Table 154: **`conference.delete`** inputs

Parameter name	Type	Description
`conferenceGUID`	string	**Required**. Globally unique identifier of the conference.
`conferenceID`	integer	Unique conference identifier. **Deprecated**. Use **`conferenceGUID`** instead.

conference.enumerate

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

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

Table 155: **`conference.enumerate`** inputs

Parameter name	Type	Description
`enumerateID`	integer	Enumerate calls may return many results so they 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`	boolean	■ true: Request only active conferences ■ false: Request all conferences. This is the default value; it is assumed if you omit the parameter. A conference is considered active, for the purpose of this filter, if one of the following conditions is true: ■ The conference has at least one participant ■ The conference has a numericID that is registered with the H.323 gatekeeper or SIP registrar Having a numeric ID alone is not enough for the conference to be considered active; if registration is disabled, a conference with a numeric ID is considered inactive unless it has at least one participant.

Table 156: **`conference.enumerate`** returned data

Parameter name	Type	Description
`conferences`	array of structs	Each member is a struct which contains all the returned information about a single conference.
`enumerateID`	string	Enumerate calls may return many results so they 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.

Table 157: **`conferences`** array struct members

Parameter name	Type	Description
`conferenceName`	string (80)	The name that refers to the conference that is the subject of your call or the response from the TelePresence Server.
`conferenceGUID`	string	Globally unique identifier of the conference.

Table 157: conferences array struct members (continued)

Parameter name	Type	Description
conferenceID	integer	Unique conference identifier. **Deprecated**. Use **conferenceGUID** instead. 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	boolean	■ true: The conference is currently active ■ false: The conference is currently inactive A conference is reported to be active if one of the following conditions is true: ■ The conference has at least one participant ■ The conference has a numericID that is registered with the H.323 gatekeeper or SIP registrar Having a numeric ID alone is not enough for the conference to be considered active; if registration is disabled, a conference with a numeric ID is considered inactive unless it has at least one participant.
persistent	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. ■ 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	boolean	Defines whether the conference is locked. Endpoints can not join a locked conference but the conference can invite them in. ■ true: The conference is locked. ■ false: The conference is not locked.
numericID	string (80)	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	boolean	Defines whether or not this item registers its **numericID** with the H.323 gatekeeper.
registerWithSIPRegistrar	boolean	Defines whether or not this item registers its **numericID** with the SIP registrar.
h239ContributionEnabled	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.

Table 157: conferences array struct members (continued)

Parameter name	Type	Description
`pin`	string (40)	The PIN for this conference. If associated with a conference, it is a string of numeric digits that must be entered to gain access to that conference. **Note**: A PIN is only valid for incoming calls - no outgoing calls will ever need to enter it. As a result of this, a conference PIN can only be set when the conference has a numeric ID. Trying to set a PIN without a numeric ID will return a fault, and clearing a conference's numeric ID will also clear that conference's PIN.

conference.invite

Invites the specified participants to the specified conference.

Avoid using the **`conferenceID`** and **`participantList`** parameters and use the replacement **`conferenceGUID`** and **`participants`** parameters instead. To identify the conference, use **`conferenceGUID`** instead of **`conferenceID`**, not both. To identify the participants, use the **`participants`** array instead of **`participantList`**, not both.

Table 158: **`conference.invite`** inputs

Parameter name	Type	Description
`conferenceGUID`	string	**Required**. Globally unique identifier of the conference.
`participants`	array of structs	**Required**. An array of structures that represent participants.
`conferenceID`	integer	Unique conference identifier. **Deprecated**. Use **`conferenceGUID`** instead.
`participantList`	string	**Deprecated**. Use **`participants`** array instead. A comma separated list of participant addresses, with optional extra information. ▪ 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` 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.

Table 159: **`participants`** array struct members

Parameter name	Type	Description
`address`	string (80)	**Required**. The address of the participant. You must prefix the address with either **`h323:`** or **`sip:`**. If you do not provide a prefix, the TelePresence Server attempts to call the address directly, using H.323 (not via the gatekeeper). The maximum length of the address is 80 characters (note that prefixes such as **`h323:`** are included in this limit). 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. The maximum length of each address is 80 characters (note that prefixes such as **`h323:`** are included in this limit). The total length of the value supplied (up to four addresses and separating commas) cannot exceed 323 characters.
`type`	string	Specifies the type of endpoint. ▪ t3: Cisco TelePresence System T3 ▪ cts: Any Cisco TelePresence System 'telepresence' endpoint (1 or 3 screen, e.g. 500, 1300, 3000) ▪ 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`	boolean	▪ true: This endpoint is conference master ▪ false: (default if omitted) This endpoint is not the conference master
`oneTableIndex`	integer	The endpoint's position if it is in a OneTable conference. Applies only if **`type`** is **`t3`**. 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`	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`	boolean	▪ 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 recording icon as appropriate. ▪ false: (default if omitted) The endpoint is a normal endpoint.

Table 159: participants array struct members (continued)

Parameter name	Type	Description
`dtmf`	string (127)	DTMF string to send after connection. The recipient(s) depends on the context of the parameter; if passed to **`conference.senddtmf`**, the sequence can be sent to one, all, or all but one of the participants. In **`conference.invite`**, you can pass it on a per participant basis. Commands that take DTMF string parameters will accept any non-DTMF ASCII characters in the string but the TelePresence Server will ignore them; it processes the string until it reaches the end, sending only the tones for characters within the set **`*#0123456789ABCD`** and pausing the tone sequence by two seconds for each comma. The TelePresence Server returns a fault if there are non-ASCII characters in the string.
`audioContentIndex`	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`**. ▪ 0: (default) The first address in the address string ▪ n-1: (maximum) The last address in a comma separated string of n addresses
`contentIndex`	integer	Defines which endpoint in a group should receive the content (if different to **`audioContentIndex`**). It is ignored unless **`audioContentIndex`** is supplied in the request.This is a zero-based index that corresponds to the entries provided in the comma separated list of endpoint addresses in **`address`**. Defaults to **`audioContentIndex`**. Use this parameter if the endpoint sending/receiving content is different to that sending/receiving audio (specified in **`audioContentIndex`**). ▪ 0: (default) The first address in the address string ▪ n-1: (maximum) The last address in a comma separated string of n addresses
`camerasCrossed`	boolean	▪ 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`	string	Overrides the aspect ratio of the layout transmitted to this participant. ▪ 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

Table 159: participants array struct members (continued)

Parameter name	Type	Description
`autoReconnect`	boolean	Defines whether the TelePresence Server attempts to re-establish the call to this endpoint (or a member of a group if the endpoint is grouped), if it fails or disconnects due to an error. ▪ True: Five retries will be attempted at intervals of 5 seconds (first retry after failure/disconnection), 15 seconds, 30 seconds, 60 seconds, 120 seconds. ▪ False: The TelePresence Server will not attempt to reconnect the call. This is the default.
`alwaysReconnect`	boolean	Defines whether the TelePresence Server should attempt to re-establish a connection to this participant in all participant-initiated disconnection scenarios. Does not apply when the TelePresence Server initiates disconnection. ▪ true: The TelePresence Server always attempts to reconnect to this endpoint, even after a deliberate disconnection from the participant. The TelePresence Server attempts to reconnect at the following intervals: 5 seconds after disconnection, 15 seconds after, 30 seconds after, 1 minute after, and 2 minutes after disconnection. If the call reconnects on any retry, then the retry schedule resets itself. ▪ false: The TelePresence Server does not always attempt to reconnect to this endpoint. It reconnects according to the description of **`autoReconnect`** instead (default).
`deferConnect`	boolean	Defines whether the TelePresence Server defers automatically connecting this participant to the conference until at least one other participant is in the conference. ▪ true: The TelePresence Server automatically connects the pre-configured participant when at least one other participant is present. ▪ false: The TelePresence Server automatically connects the pre-configured participant as soon as the conference starts (default).
`autoDisconnect`	boolean	Defines whether the call will automatically disconnect fron the conference when other particpants disconnect. ▪ true: The call automatically disconnects when the other participants disconnect. ▪ false: The call does not automatically disconnect when other participants disconnect (default).
`defaultLayoutSingleScreen`	string	One of **`single`**, **`activePresence`**, **`equal`**, or **`prominent`**. Defines which layout should be displayed on the participant's endpoint if it is a single-screen endpoint. This parameter is ignored if the participant is using a multiscreen endpoint.
`defaultLayoutMultiScreen`	string	One of **`single`** or **`activePresence`**. Defines which layout should be displayed on the participant's endpoint if it is a multiscreen endpoint. This parameter is ignored if the participant is using a single-screen endpoint.

Table 159: participants array struct members (continued)

Parameter name	Type	Description
forceDefaultLayout	boolean	Defines whether the layout sent to the participant is forced to the default for their endpoint type, or whether the user may change the layout. ■ true: The layout is forced to be the default for the user's endpoint type (the value of either **defaultLayoutSingleScreen** or **defaultLayoutMultiScreen**). The user cannot change the layout. ■ false: The layout is not forced; the user may change the layout if the endpoint is capable.
automaticGainControl	boolean	Defines whether automatic gain control is enabled. If not specified, the conference default is used.
allowStarSixMuting	boolean	Defines whether participants can mute their audio by pressing *6. **true** allows the participant to use the *6 combination to mute/unmute. If not specified, the TelePresence Server's default value is used for the participant. If the TelePresence Server's default is not specified (via the web UI), then it defaults to **true**.

Table 160: **conference.invite** returned data

Parameter name	Type	Description
participantList	array of structs	Array of participants. Each member of the array is a struct that represents a participant on the TelePresence Server.

participantList array

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.

Table 161: **participantList** array struct members

Parameter name	Type	Description
participantGUID	string	The GUID of this participant, assigned by the TelePresence Server.
participantID	integer	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	string	The address of the participant. These addresses are as you supplied them in the **participants** array, to make them easier to compare.
groupAddressList	array of strings	Each member of the array is an address of one of the group members. This array is only returned for endpoint groups; that is, when the **address** of the particpant in the conference.invite **participants** array was set to a comma-separated list of addresses. The index position of each endpoint's address corresponds with the position in the comma-separated list provided in the **address** parameter.

conference.senddtmf

Sends a DTMF sequence to some or all participants in the specified conference. You must specify the conference and the DTMF string (up to 50 characters). To identify the conference, use **`conferenceGUID`** instead of **`conferenceID`**, not both.

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

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.

Table 162: **`conference.senddtmf`** inputs

Parameter name	Type	Description
`conferenceGUID`	string	**Required**. Globally unique identifier of the conference.
`conferenceID`	integer	Unique conference identifier. **Deprecated**. Use **`conferenceGUID`** instead.
`dtmf`	string (127)	**Required**. DTMF string to send after connection. The sequence can be sent to one, all, or all but one, of the participants in the specified conference. Commands that take DTMF string parameters will accept any non-DTMF ASCII characters in the string but the TelePresence Server will ignore them; it processes the string until it reaches the end, sending only the tones for characters within the set **`*#0123456789ABCD`** and pausing the tone sequence by two seconds for each comma. The TelePresence Server returns a fault if there are non-ASCII characters in the string.
`participantGUID`	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`	integer	The unique ID of this participant, assigned by the TelePresence Server. **Deprecated**. Use **`participantGUID`** instead.
`omitGUID`	string	A participant GUID. Prevents the 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 in this command, **`omitGUID`** is ignored.
`omitID`	integer	**Deprecated**. Use **`omitGUID`** instead. A **`participantID`**. Prevents this participant from receiving the DTMF string specified in **`dtmf`**.

conference.sendmessage

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

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

If you choose to specify a participant, the message will only go to that participant. To identify a participant, use the **`participantGUID`** instead of **`participantID`**, not both.

Table 163: **`conference.sendmessage`** inputs

Parameter name	Type	Description
`conferenceGUID`	string	**Required**. Globally unique identifier of the conference.
`conferenceID`	integer	Unique conference identifier. **Deprecated**. Use **`conferenceGUID`** instead.
`message`	string (500)	**Required**. Message to send to conference.
`participantGUID`	string	The GUID of this participant, assigned by the TelePresence Server.
`participantID`	integer	The unique ID of this participant, assigned by the TelePresence Server. **Deprecated**. Use **`participantGUID`** instead.
`position`	integer	Defines where the message displays on the layout. ▪ `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. ▪ `7`, `8`, or `9`: The message displays near the bottom of the layout; aligned to the left, center, or right respectively.
`duration`	integer (>0)	Period of time (in seconds) for which the message is displayed to participants. The TelePresence Server will accept `0` but the behavior is undefined in this case. Default is `30`.

conference.sendwarning

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

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

Table 164: **`conference.sendwarning`** inputs

Parameter name	Type	Description
`conferenceGUID`	string	**Required**. Globally unique identifier of the conference.
`conferenceID`	integer	Unique conference identifier. **Deprecated**. Use **`conferenceGUID`** instead.
`secondsRemaining`	integer	The number of seconds from now in which the conference will end. This value is used when informing CTS endpoints (using XCCP) that the conference is ending.

conference.set

Edit the configuration of the specified conference.

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

Table 165: **`conference.set`** inputs

Parameter name	Type	Description
`conferenceGUID`	string	**Required**. Globally unique identifier of the conference.
`conferenceID`	integer	Unique conference identifier. **Deprecated**. Use **`conferenceGUID`** instead.
`numericID`	string (80)	Used for registration with H.323 gatekeeper / SIP registrar, and to dial in to the conference.
`registerWithGatekeeper`	boolean	Defines whether or not this item registers its **`numericID`** with the H.323 gatekeeper.
`registerWithSIPRegistrar`	boolean	Defines whether or not this item registers its **`numericID`** with the SIP registrar.
`roundTableEnable`	boolean	Defines whether the conference is in round table mode. **Deprecated**. Use **`OneTableMode`** instead. If you supply both **`roundTableEnable`** and **`OneTableMode`**, then the TelePresence Server will use **`OneTableMode`** without returning an error.
`oneTableMode`	integer	To set up one table mode, use **`oneTableMode`** instead of **`roundTableEnable`**, not both. ■ 0: OneTableMode off ■ 1: 4 person OneTableMode
`h239ContributionEnabled`	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`	boolean	Defines whether the conference is locked. Endpoints can not join a locked conference but the conference can invite them in. ■ true: The conference is locked. ■ false: The conference is not locked.
`lockDuration`	integer	The period of time (in seconds) from now until the conference lock expires. Requires that **`locked`** is **`true`** and ignored otherwise.
`duration`	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`**.

Table 165: conference.set inputs (continued)

Parameter name	Type	Description
audioPortLimitSet	boolean	Defines whether the **audioPortLimit** is applied. You **must** provide an **audioPortLimit** if you set **audioPortLimitSet** to **true**. If you set it **false**, the call clears the existing **audioPortLimit**. ▪ true: Limits the number of audio ports to the value in **audioPortLimit** ▪ false: **audioPortLimit** is ignored if it is present
audioPortLimit	integer	The limit on the number of audio ports this conference may allow.
videoPortLimitSet	boolean	Defines whether the **videoPortLimit** is applied. You **must** provide a **videoPortLimit** if you set **videoPortLimitSet** to **true**. If you set it **false**, the call clears the existing **videoPortLimit**. ▪ true: Limits the number of video ports to the value in **videoPortLimit** ▪ false: **videoPortLimit** is ignored if it is present
videoPortLimit	integer	The limit on the number of video ports this conference may allow.
useLobbyScreen	boolean	Defines whether the conference shows the lobby screen.
lobbyMessage	string (500)	The lobby screen message.
useWarning	boolean	Defines whether the conference sends 'This conference is about to end' warning.
automaticGainControl	boolean	Defines whether automatic gain control is enabled. If not specified, the conference default is used.
encryptionRequired	boolean	Defines whether encryption is required for this conference. ▪ true: Encryption is required. ▪ false: Encryption is optional (default).
pin	string (40)	The PIN for this conference. If associated with a conference, it is a string of numeric digits that must be entered to gain access to that conference. **Note:** A PIN is only valid for incoming calls - no outgoing calls will ever need to enter it. As a result of this, a conference PIN can only be set when the conference has a numeric ID. Trying to set a PIN without a numeric ID will return a fault, and clearing a conference's numeric ID will also clear that conference's PIN.

conference.status

Reports the current status of the specified conference and its participants. To identify the conference, use **conferenceGUID** instead of **conferenceID**, not both.

Table 166: **`conference.status`** inputs

Parameter name	Type	Description
`conferenceGUID`	string	**Required**. Globally unique identifier of the conference.
`conferenceID`	integer	Unique conference identifier. **Deprecated**. Use **`conferenceGUID`** instead.
`enumerateID`	string	Enumerate calls may return many results so they may include this parameter in the response. If the response includes an enumerateID, your application should pass the ID to a 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 your application omits the enumerateID, the TelePresence Server will start a new enumeration and return the first set of results.

Table 167: **`conference.status`** returned data

Parameter name	Type	Description
`conferenceGUID`	string	Globally unique identifier of the conference.
`conferenceID`	integer	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.
`enumerateID`	string	Only returned if there is more data to return than can be contained in one 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.
`active`	boolean	■ true: The conference is currently active ■ false: The conference is currently inactive A conference is reported to be active if one of the following conditions is true: ■ The conference has at least one participant ■ The conference has a numericID that is registered with the H.323 gatekeeper or SIP registrar Having a numeric ID alone is not enough for the conference to be considered active; if registration is disabled, a conference with a numeric ID is considered inactive unless it has at least one participant.

Table 167: conference.status returned data (continued)

Parameter name	Type	Description
`persistent`	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. ■ 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`	integer	Period of time (in seconds) until the conference ends and is deleted. This parameter is not allowed if `persistent` is `true`.
`locked`	boolean	Defines whether the conference is locked. Endpoints can not join a locked conference but the conference can invite them in. ■ true: The conference is locked. ■ false: The conference is not locked.
`lockDuration`	integer	The period of time (in seconds) from now until the conference lock expires. Requires that `locked` is `true` and ignored otherwise.
`roundTableEnable`	boolean	Defines whether the conference is in round table mode. **Deprecated**. Use `OneTableMode` instead. 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`	integer	■ 0: OneTableMode off ■ 1: 4 person OneTableMode
`h239ContributionID`	integer	The `participantID` of the endpoint that is contributing H.239 content. Zero if there is no H.239 contribution.
`portsVideoFree`	integer	Count of the currently unused video ports. Zero if the conference is inactive.
`portsAudioFree`	integer	Count of the currently unused audio ports. Zero if the conference is inactive.
`portsContentFree`	integer	Count of the currently unused content ports. Zero if the conference is inactive.
`numericID`	string (80)	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.

Table 167: conference.status returned data (continued)

Parameter name	Type	Description
`pin`	string (40)	The PIN for this conference. If associated with a conference, it is a string of numeric digits that must be entered to gain access to that conference. **Note**: A PIN is only valid for incoming callsâ€"no outgoing calls will ever need to enter it. As a result of this, a conference PIN can only be set when the conference has a numeric ID. Trying to set a PIN without a numeric ID will return a fault, and clearing a conferenceâ€™s numeric ID will also clear that conferenceâ€™s PIN.
`registerWithGatekeeper`	boolean	Defines whether or not this item registers its **`numericID`** with the H.323 gatekeeper.
`registerWithSIPRegistrar`	boolean	Defines whether or not this item registers its **`numericID`** with the SIP registrar.
`recording`	boolean	True if this conference is being recorded by a recording device specified in **`conference.invite`**.
`audioPortLimitSet`	boolean	Defines whether the **`audioPortLimit`** is applied. ■ true: Limits the number of audio ports to the value in **`audioPortLimit`** ■ false: **`audioPortLimit`** is ignored if it is present
`audioPortLimit`	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`	boolean	Defines whether the **`videoPortLimit`** is applied. ■ true: Limits the number of video ports to the value in **`videoPortLimit`** ■ false: **`videoPortLimit`** is ignored if it is present
`videoPortLimit`	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`**.
`automaticGainControl`	boolean	Defines whether automatic gain control is enabled. If not specified, the conference default is used.
`encryptionRequired`	boolean	Defines whether encryption is required for this conference. ■ true: Encryption is required. ■ false: Encryption is optional (default).
`participantList`	array of structs	Array of participants. Each member of the array is a struct that represents a participant on the TelePresence Server.

`participantList` array

The returned **`participantList`** is an array of the conference's current and previous participants. The member structs of this array contain a different set of parameters than those returned in the **`participantList`** of a **`conference.invite`** call.

The array will contain all current and previous participants, up to the TelePresence Server's global maximum of 208, unless the previous participants are deleted via the API or cleared via the UI. The array can be returned empty if there are no current participants and if all previous participants were cleared.

Table 168: **`participantList`** array struct members

Parameter name	Type	Description
`participantGUID`	string	The GUID of this participant, assigned by the TelePresence Server.
`participantID`	integer	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`	integer	State of the call between the TelePresence Server and this participant. ▪ 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`	integer	**Deprecated:** use **`endpointCategory`** instead. 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. ▪ 1: Normal endpoint ▪ 3: Grouped endpoints ▪ 4: T3 ▪ 5: Cisco CTS or other TIP capable endpoints
`endpointCategory`	string	▪ normal: Normal endpoint ▪ group: Grouped endpoints ▪ t3: T3 ▪ cts: Cisco CTS or other TIP capable endpoints
`callStartMute`	boolean	True if this endpoint is being sent black video during call setup.
`master`	boolean	▪ true: This endpoint is conference master ▪ false: (default if omitted) This endpoint is not the conference master
`callType`	string	▪ audio: An audio only participant ▪ video: A video participant
`callProtocol`	string	▪ sip: This call uses the SIP protocol. ▪ h323: This call uses the H.323 protocol.

Table 168: participantList array struct members (continued)

Parameter name	Type	Description
disconnectReason	string	The reason why the endpoint disconnected. Only returned for disconnected participants. ■ unspecified: Unspecified error ■ localTeardown: Requested by administrator ■ noAnswer: No answer ■ rejected: Call rejected ■ busy: Busy ■ gatekeeperError: Gatekeeper error ■ remoteTeardown: Left conference ■ timeout: Call timed out ■ protocolError: Protocol error ■ unreachable: Endpoint is unreachable ■ networkError: Network error ■ capabilityNegotiationError: Capability negotiation error ■ dnsFailure: DNS failure ■ noMediaReceived: The TelePresence Server disconnected the call because the endpoint was unexpectedly not sending media for at least 30 seconds.
rxPreviewURL	string	The URL to retrieve a jpeg snapshot of video received from this participant. Only returned for active participants.
txPreviewURL	string	The URL to retrieve a jpeg snapshot of video sent to this participant. Only returned for active participants.
callDuration	integer	The duration of the call in seconds. Only returned for active participants.
callDirection	string	Only returned for active participants. This parameter is not present if **callState** is **0** (not connected). ■ incoming: The participant called in to the TelePresence Server ■ outgoing: The TelePresence Server called out to the participant
callBandwidth	integer	The Tx (transmit) bandwidth negotiated from the TelePresence Server to this participant (in kbps). Only returned for active participants.
micMute	boolean	True if far end microphone is muted. Only returned for active participants.
recordingDevice	boolean	■ 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.

Table 168: participantList array struct members (continued)

Parameter name	Type	Description
txAudioMute	boolean	Defines whether the TelePresence Server mutes the audio signal transmitted to this endpoint. Only returned for active participants.
rxAudioMute	boolean	Defines whether the TelePresence Server mutes the audio signal received from this endpoint. Only returned for active participants.
txVideoMute	boolean	Defines whether the TelePresence Server mutes the video signal transmitted to this endpoint. Only returned for active participants.
rxVideoMute	boolean	Defines whether the TelePresence Server mutes the video signal received from this endpoint. Only returned for active participants.
isImportant	boolean	Defines whether the participant is important (i.e. the participant's transmitted video is given preference over others when composing video). ■ 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.
groupAddressList	array of strings	Each member of the array is an address of one of the group members. This array is only returned for endpoint groups; that is, when the **address** of the particpant in the conference.invite **participants** array was set to a comma-separated list of addresses. The index position of each endpoint's address corresponds with the position in the comma-separated list provided in the **address** parameter.
groupCallStateList	array of integers	This array is only returned for endpoint groups. Each member of the array is an integer that represents the state of the call between one of the group members and the TelePresence Server. The index position of the endpoint's call state integer corresponds to the index position of the endpoint's address in **groupAddressList**. ■ 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

Table 168: participantList array struct members (continued)

Parameter name	Type	Description
groupAudioIndex	integer	This parameter is only returned for endpoint groups. It is the index of the endpoint in the group that is nominated to receive the conference audio channel. This is a zero-based index that corresponds to the entries returned in **groupAddressList**. ■ 0: (default) The endpoint whose address is first in **groupAddressList** is nominated to receive audio. ■ n-1: (maximum) The endpoint whose address is last in **groupAddressList** is nominated to receive audio.
groupContentIndex	integer	This parameter is only returned for endpoint groups. It is the index of the endpoint in the group that is nominated to receive the conference content channel. This is a zero-based index that corresponds to the entries returned in **groupAddressList**. ■ 0: (default) The endpoint whose address is first in **groupAddressList** is nominated to receive content. ■ n-1: (maximum) The endpoint whose address is last in **groupAddressList** is nominated to receive content.

conference.uninvite

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.

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

To identify participants to uninvite, **use only one of the following: participantListGUID**, **participantList**, or **participantListID**.

Table 169: **conference.uninvite** inputs

Parameter name	Type	Description
conferenceGUID	string	**Required**. Globally unique identifier of the conference.
conferenceID	integer	Unique conference identifier. Required if **conferenceGUID** is omitted. **Deprecated**. Use **conferenceGUID** instead.
participantListGUID	string	**Required**. 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	string	A comma separated list of participant addresses. used to identify which participants to remove from the conference. Example string: **10.2.171.232, 10.47.2.246, h323:numericID@domain.com** Required if **participantListGUID** and **participantListID** are omitted.

Table 169: conference.uninvite inputs (continued)

Parameter name	Type	Description
participantListID	string	Comma separated list of **participantID**s that identifies which participants to remove from the conference. For example; **1024, 1056**. Required if **participantListGUID** and **participantList** are omitted. **Deprecated**. Use **participantListGUID** instead.

device.feature.add

Adds a license or feature to the TelePresence Server. You need to obtain a key from Cisco or one of its resellers prior to running this command.

Table 170: **device.feature.add** inputs

Parameter name	Type	Description
key	string	**Required**. Use this unique code when you wish to add conferencing capacity or an optional feature to your TelePresence Server.

device.feature.remove

Removes a license or feature from the TelePresence Server. Use **device.query** to read the keys from a TelePresence Server.

Table 171: **device.feature.remove** inputs

Parameter name	Type	Description
key	string	**Required**. The unique code associated with the optional feature or license that you wish to remove from the TelePresence Server.

device.query

Returns high level status information about the device. This command takes no input parameters.

Table 172: **device.query** returned data

Parameter name	Type	Description
currentTime	dateTime.iso8601	The system's current date and time.
restartTime	dateTime.iso8601	The system's date and time when it started.
uptime	integer	The difference, in seconds, between the system's current time and the system's restart time.
serial	string	Serial number of this device.

Table 172: device.query returned data (continued)

Parameter name	Type	Description
apiVersion	string	Version number of the API implemented by this TelePresence Server.
activatedFeatures	array of structs	Each member of the array is a struct, representing an active feature. See Table 173: Active feature struct members [p.145].
activatedLicenses	array of structs	Each member of the array is a struct, representing an active license. See Table 174: Active license struct members [p.145].
shutdownStatus	string	Displays one of the following: **notShutdown**, **shutdownInProgress**, **shutdown**, or **error**.
mediaResourceRestarts	integer	The count of unexpected restarts that have occurred on the device's media resources (signal processor chips).
portsVideoTotal	integer	The total number of video ports enabled by the screen licenses on this TelePresence Server. If there are 10 screen licenses, then **videoPortsTotal** will be 10 if the unit is in FullHD mode, or 20 if the unit is in HD mode. If the unit is the master in a cluster, then **portsVideoTotal** reports the total number of video ports enabled by the screen licenses applied across the whole cluster.
portsAudioTotal	integer	The total number of audio-only ports enabled by the screen licenses on this TelePresence Server. If the unit is the master in a cluster, then **portsAudioTotal** reports the total number of audio-only ports enabled by the screen licenses applied across the whole cluster.

Table 173: Active feature struct members

Parameter name	Type	Description
feature	string	The name of the feature, eg. **Encryption**.
key	string	The unique code associated with the feature.
expiry	dateTime.iso8601	The time at which this temporary key will expire. **expiry** is not present for permanent keys.

Table 174: Active license struct members

Parameter name	Type	Description
license	string	The name of the license.
ports	integer	The number of screen licenses provided by this license.
key	string	The unique code associated with the license.
expiry	dateTime.iso8601	The time at which this temporary key will expire. **expiry** is not present for permanent keys.

device.network.query

Queries the device for its network information. The call takes no parameters and returns the following data structures. Some of the data listed below will be omitted if the interface is not enabled or configured. The query returns empty strings or dashes for addresses that are not configured.

Note: Packet counts and other statistics are measured with 32-bit signed integers, and may therefore wrap.

Table 175: `device.network.query` returned data

Parameter name	Type	Description
`portA`	struct	The struct contains configuration and status information for Ethernet port A on the device. See Table 176: Port struct members [p.146].
`dns`	array of structs	Each member of the array is a struct representing a set of DNS parameters for the queried device. See Table 177: DNS struct members [p.147].

Table 176: Port struct members

Parameter name	Type	Description
`enabled`	boolean	Whether the port is enabled.
`ipv4Enabled`	boolean	Whether IPv4 interface is enabled. Always returned unless there are no IP interfaces enabled on the port (neither IPv4 nor IPv6 is enabled).
`ipv6Enabled`	boolean	Whether IPv6 interface is enabled. Always returned unless there are no IP interfaces enabled on the port (neither IPv4 nor IPv6 is enabled).
`linkStatus`	boolean	Whether the Ethernet connection to this port is active.
`speed`	integer	Speed of the connection on this Ethernet port. One of 10, 100 or 1000, in Mbps.
`fullDuplex`	boolean	Whether the port can support a full-duplex connection.
`macAddress`	string	MAC address of this port. A 12-character string of hex digits with no separators.
`packetsSent`	integer	Number of packets sent from this Ethernet port.
`packetsReceived`	integer	Number of packets received on this Ethernet port.
`multicastPacketsSent`	integer	Number of multicast packets sent from this Ethernet port.
`multicastPacketsReceived`	integer	Number of multicast packets received on this Ethernet port.
`bytesSent`	integer	Number of bytes sent by the device.
`bytesReceived`	integer	Number of bytes received by the device.
`queueDrops`	integer	Number of packets dropped from the queue on this network port.

Table 176: Port struct members (continued)

Parameter name	Type	Description
`collisions`	integer	Count of the network collisions recorded by the device.
`transmitErrors`	integer	Count of transmission errors on this Ethernet port.
`receiveErrors`	integer	Count of receive errors on this port.
`bytesSent64`	string	64-bit versions of the **`bytesSent`** statistic expressed as a string rather than an integer.
`bytesReceived64`	string	64-bit versions of the **`bytesReceived`** statistic expressed as a string rather than an integer.
`dhcpv4`	boolean	Whether the ipv4 address is allocated by DHCP. Not returned if not configured.
`ipv4Address`	string	IPv4 address in the dotted quad format. Not returned if not configured.
`ipv4SubnetMask`	string	IPv4 subnet mask in the dotted quad format. Not returned if not configured.
`defaultipv4Gateway`	string	IPv4 address in the dotted quad format. Not returned if not configured.
`ipv6Address`	string	IPv6 address in CIDRformat. Not returned if not configured.
`ipv6Conf`	string	Indicates how the IPv6 address is assigned. One of **`automatic`** (IPv6 address is configured by SLAAC/DHCPv6) or **`manual`** (IPv6 address is configured manually). Not returned if not configured.
`ipv6PrefixLength`	integer	Length of the IPv6 address prefix. Not returned if not configured.
`defaultIpv6Gateway`	string	Address of the IPv6 default gateway in CIDR format. Not returned if not configured.
`linkLocalIpv6Address`	string	Link local IPv6 address in CIDR format. Not returned if not configured.
`linkLocalIpv6PrefixLength`	integer	Length of the link local IPv6 address prefix. Not returned if not configured.

Table 177: DNS struct members

Parameter name	Type	Description
`hostName`	string	Host name of the queried device.
`nameServer`	string	IP address of the name server, in dotted quad format (IPv4) or CIDR format (IPv6).
`nameServerSecondary`	string	IP address of the secondary name server, in dotted quad format (IPv4) or CIDR format (IPv6).
`domainName`	string	Domain name of the queried device (DNS suffix).

device.health.query

Returns the current status of the device, such as health monitors and CPU load. This command takes no input parameters.

Table 178: **`device.health.query`** returned data

Parameter name	Type	Description
`cpuLoad`	integer	CPU load expressed as a percentage of the maximum.
`fanStatus`	string	**`ok`** or **`outOfSpec`**. This parameter is returned only on appliances, eg. Media 310 or TelePresence Server 7010, which have their own fans. This parameter is not returned for TelePresence Server blades.
`fanStatusWorst`	string	Worst fan status recorded on this device since it restarted. One of **`ok`** or **`outOfSpec`**. This parameter is returned only on appliances, eg. Media 310 or TelePresence Server 7010, which have their own fans. This parameter is not returned for TelePresence Server blades.
`temperatureStatus`	string	One of **`ok`** (the temperature is currently within the normal operating range), **`outOfSpec`** (the temperature is currently outside the normal operating range), or **`critical`** (the temperature is too high and the device will shutdown if this condition persists).
`temperatureStatusWorst`	string	Worst temperature status recorded on this device since it booted. One of **`ok`**, **`outOfSpec`**, or **`critical`**.
`rtcBatteryStatus`	string	Current status of the RTC battery (Real Time Clock). One of **`ok`**, **`outOfSpec`**, or **`critical`**.
`rtcBatteryStatusWorst`	string	Worst status of the RTC battery (Real Time Clock) recorded on this device since it booted. One of **`ok`**, **`outOfSpec`**, or **`critical`**.
`voltagesStatus`	string	One of **`ok`** (the voltage is currently within the normal range), **`outOfSpec`** (the voltage is currently outside the normal range), or **`critical`**.
`voltagesStatusWorst`	string	Worst voltage status recorded on this device since it booted. One of **`ok`**, **`outOfSpec`**, or **`critical`**.
`operationalStatus`	string	One of **`active`** (the device is active), **`shuttingDown`** (the device is shutting down), **`shutDown`** (the device has shut down), or **`unknown`**.

device.restartlog.query

Returns the restart log - also known as the system log on the web interface. This command takes no input parameters.

Table 179: **`device.restartlog.query`** returned data

Parameter name	Type	Description
`log`	array of structs	Each member of the array is a struct containing a restart **`reason`**. See Table 180: Log struct members [p.149]. This information source is called "system log" in the web interface.

Table 180: Log struct members

Parameter name	Type	Description
`time`	dateTime.iso8601	Date and time of the restart.
`reason`	string	Reason for the device restart. See Table 181: Restart reason enumerated type [p.149].

Table 181: Restart **`reason`** enumerated type

`reason` value	Description
User requested shutdown	The device restarted normally after a user initiated a shutdown.
User requested reboot from web interface	The device restarted itself because a user initiated a reboot via the web interface.
User requested upgrade	The device restarted itself because a user initiated an upgrade.
User requested reboot from console	The device restarted itself because a user initiated a reboot via the console.
User requested reboot from API	The device restarted itself because a user initiated a reboot via the API.
User requested reboot from FTP	The device restarted itself because a user initiated a reboot via FTP.
User requested shutdown from supervisor	The device restarted normally after a user initiated a shutdown from the supervisor.
User requested reboot from supervisor	The device restarted itself because a user initiated a reboot via the supervisor.
User reset configuration	The device restarted itself because a user reset the configuration.
Cold boot	The device restarted itself because a user initiated a cold boot.
unknown	The software is unaware why the device restarted.

device.restart

Restarts the device, or shuts it down without a restart. This command does not return any parameters.

Table 182: **`device.restart`** input parameters

Parameter name	Type	Description
`shutdownOnly`	boolean	(Optional) Set to **`true`** to shut down without restarting. Default: **`false`**.

feedbackReceiver.configure

Configures the device to send feedback about the specified **`subscribedEvents`** to the specified **`receiverURI`**.

Table 183: **`feedbackReceiver.configure`** inputs

Parameter name	Type	Description
`receiverURI`	string (255)	**Required**. Fully-qualified **`http`** or **`https`** URI (for example, **`http://tms1:8080/RPC2`**) to which feedback events are sent. If no port number is specified, the device uses the protocol defaults (80 and 443 respectively).
`receiverIndex`	integer (< 0, or 1–20 inclusive)	Index of the feedback receiver indicating the slot that this receiver should use. A negative value indicates that the feedback receiver should use any available slot (preferred). Default: 1. **Note:** The default **`receiverIndex`** is 1, and will always overwrite a feedback receiver in the first index position. You should query the device first, or use a negative value, if you want to be certain not to overwrite an existing feedback receiver.
`sourceIdentifier`	string (255) ASCII characters only	Identifier string for the receiver. The originating device uses this parameter to identify itself to the listening receiver (or receivers). If the parameter is not explicitly set, the device identifies itself with the MAC address of its Ethernet port A interface. Default: empty.
`subscribedEvents`	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. See Feedback events [p.117]. If this array is absent, the receiver subscribes to all notifications by default. Default: all events.

Table 184: **`feedbackReceiver.configure`** returned data

Parameter name	Type	Description
`receiverIndex`	integer	Position of this feedback receiver in the device's table of feedback receivers.

feedbackReceiver.query

Requests a list of all the feedback receivers that have previously been configured for the device. It does not accept parameters other than the authentication strings. If there are no feedback receivers to enumerate, **`feedbackReceiver.query`** returns an empty **`receivers`** array.

Table 185: **`feedbackReceiver.query`** returned data

Parameter name	Type	Description
`receivers`	array	Array of feedback receivers, with members corresponding to the entries in the receivers table on the web interface of the device.

Table 186: Feedback receiver struct members

Parameter name	Type	Description
`index`	integer (1–20)	Position of this feedback receiver in the table of feedback receivers. The index number is also the feedback receiver ID.

Table 186: Feedback receiver struct members (continued)

Parameter name	Type	Description
`sourceIdentifier`	string (255) ASCII characters only	Source identifier string, which can be empty. The originating device uses this parameter to identify itself to the listening receiver (or receivers). If the parameter is not explicitly set, the device identifies itself with the MAC address of its Ethernet port A interface.
`receiverURI`	string (255)	Fully-qualified **`http`** or **`https`** URI (for example, **`http://tms1:8080/RPC2`**) to which feedback events are sent.

feedbackReceiver.reconfigure

Overwrites the configuration of an existing feedback receiver with any parameters that you supply. The TelePresence Server keeps the current configuration for any parameters that you do not specify.

Table 187: **`feedbackReceiver.reconfigure`** inputs

Parameter name	Type	Description
`receiverIndex`	integer (1–20)	**Required**. Index of the feedback receiver to be reconfigured. The call returns a fault if there is no feedback receiver at the specified **`receiverIndex`**.
`receiverURI`	string (255)	Fully-qualified **`http`** or **`https`** URI (for example, **`http://tms1:8080/RPC2`**) to which feedback events are sent. If omitted, the device uses the originally configured **`receiverURI`**.
`sourceIdentifier`	string (255) ASCII characters only	Identifier string for the receiver. The originating device uses this parameter to identify itself to the listening receiver (or receivers). If omitted, the device uses the originally configured **`sourceIdentifier`**.
`subscribedEvents`	array	Array of strings identifying the events to which the receiver subscribes. See Feedback events [p.117]. If omitted, the event notifications set in the original configuration request remain unchanged.

feedbackReceiver.remove

Removes the specified feedback receiver. This command returns no data.

Table 188: **`feedbackReceiver.remove`** inputs

Parameter name	Type	Description
`receiverIndex`	integer (1–20)	**Required**. Index of the feedback receiver to be removed.

feedbackReceiver.status

Asks the device for a list of all the events to which a feedback receiver subscribes.

Table 189: **`feedbackReceiver.status`** inputs

Parameter name	Type	Description
`receiverIndex`	integer (1–20)	**Required**. Index of the feedback receiver.

Table 190: **`feedbackReceiver.status`** returned data

Parameter name	Type	Description
`receiverIndex`	integer (1–20)	Index of the feedback receiver entry, which also serves as the feedback receiver ID.
`sourceIdentifier`	string (255) ASCII characters only	Identifier string for the receiver. The originating device uses this parameter to identify itself to the listening receiver (or receivers). If the parameter is not explicitly set, the device identifies itself with the MAC address of its Ethernet port A interface.
`receiverURI`	string (255)	Fully-qualified **`http`** or **`https`** URI (for example, **`http://tms1:8080/RPC2`**) to which feedback events are sent.
`subscribedEvents`	array	Array of strings identifying the event names that are enabled for this feedback receiver. See Feedback events [p.117].

participant.diagnostics

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 (an XML-RPC methodCall with methodName **`participantDiagnosticsResponse`**) on the listening interface.

The TelePresence Server can handle up to 10 concurrent asynchronous requests of this type, so this command may fail with fault code 203 if the number of pending requests exceeds this limit.

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 [p.163]

Each of the diagnostics arrays may contain zero or more stream structs. If there are no streams of a particular type, the corresponding array is returned empty. If there are stream structs in the diagnostics array, then the stream struct will contain relevant parameter/value pairs from Table 193: Diagnostics arrays' struct members [p.153].

Table 191: **`participant.diagnostics`** inputs

Parameter name	Type	Description
`participantGUID`	string	**Required**. The GUID of this participant, assigned by the TelePresence Server.
`receiverURI`	string (255)	**Required**. 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).

Table 191: participant.diagnostics inputs (continued)

Parameter name	Type	Description
`sourceIdentifier`	string (255 ASCII)	Identifier string for the receiver. The originating device uses this parameter to identify itself to the listening receiver (or receivers). If the parameter is not explicitly set, the device identifies itself with the MAC address of its Ethernet port A interface. Default: empty.

Table 192: **`participant.diagnostics`** returned data

Parameter name	Type	Description
`participantGUID`	string	The GUID of this participant, assigned by the TelePresence Server.
`sourceIdentifier`	string (255 ASCII)	Identifier string for the receiver. The originating device uses this parameter to identify itself to the listening receiver (or receivers). If the parameter is not explicitly set, the device identifies itself with the MAC address of its Ethernet port A interface. Default: empty.
`audioRx`	array of structs	Each member of the array is a struct which represents an audio stream received from the participant's endpoint.
`audioTx`	array of structs	Each member of the array is a struct which represents an audio stream transmitted to the participant's endpoint.
`auxiliaryAudioRx`	array of structs	Each member of the array is a struct which represents an auxiliary audio stream received from the participant's endpoint.
`auxiliaryAudioTx`	array of structs	Each member of the array is a struct which represents an auxiliary audio stream transmitted to the participant's endpoint.
`videoRx`	array of structs	Each member of the array is a struct which represents a video stream received from the participant's endpoint.
`videoTx`	array of structs	Each member of the array is a struct which represents a video stream transmitted to the participant's endpoint.
`contentVideoRx`	array of structs	Each member of the array is a struct which represents a content video stream received from the participant's endpoint.
`contentVideoTx`	array of structs	Each member of the array is a struct which represents a content video stream transmitted to the participant's endpoint.

Table 193: Diagnostics arrays' struct members

Parameter name	Type	Description
`codec`	string	Present for all stream types. The codec in use, or **`other`** for undefined codecs.
`encrypted`	boolean	Present for all stream types. True if the stream data is encrypted.

Table 193: Diagnostics arrays' struct members (continued)

Parameter name	Type	Description
muted	boolean	Present for all stream types except the content channel (tx and rx). True if the stream is muted.
channelBitRate	integer	Present for all stream types. Bit rate of the channel in bits per second (bps).
packetsSent	integer	Count of packets sent in this stream.
packetsReceived	integer	Count of packets received in this stream.
packetErrors	integer	Count of packets with errors in this stream.
packetsMissing	integer	Count of packets missing from this stream.
framesReceived	integer	Count of frames received in this stream.
frameErrors	integer	Count of frames with errors in this stream.
jitter	integer	Current jitter in this stream, measured in milliseconds (ms).
energy	integer	The level of the signal, supplied in decibels (dB).
configuredBitRate	integer	The configured bit rate of this stream, in bits per second (bps).
configuredBitRateReason	string	■ 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	integer	The expected bit rate of this stream, in bits per second (bps).
expectedBitRateReason	string	■ 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	integer	The measured bit rate of this stream, in bits per second (bps).
frameRate	integer	The frame rate of the video stream, in frames per second (fps).
fastUpdateRequestsSent	integer	The count of fast update requests sent in this stream.
fastUpdateRequestsReceived	integer	The count of fast update requests received in this stream.
packetsLost	integer	The number of packets lost from this stream, as reported by RTCP from the far end.
clearPathOverhead	integer	Only returned if ClearPath has been negotiated. The percentage of FEC overhead in this media stream. The value 50, for example, means that one FEC packet is used to protect every two media packets.

Table 193: Diagnostics arrays' struct members (continued)

Parameter name	Type	Description
`clearPathRecovered`	integer	Only returned if ClearPath has been negotiated. The number of media packets recovered using FEC.
`clearPathLTRF`	boolean	Only returned if ClearPath has been negotiated. **`true`** if long-term reference frames are being inserted in this stream.
`clearPathLTRFRepaired`	integer	Only returned if ClearPath has been negotiated. The number of frames repaired by referencing the long-term reference frames embedded in this stream.

participant.enumerate

Returns an array of the participants on the queried TelePresence Server.

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

Table 194: **`participant.enumerate`** inputs

Parameter name	Type	Description
`enumerateID`	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.

Table 195: **`participant.enumerate`** returned data

Parameter name	Type	Description
`participants`	array of structs	Each member of the array is a struct that represents a single participant.
`enumerateID`	string	Enumerate calls may return many results so they may include this parameter in the response. If the response includes an enumerateID, the application should pass the ID to a 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.

Table 196: **`participants`** array struct members

Parameter name	Type	Description
`participantGUID`	string	The GUID of this participant, assigned by the TelePresence Server.
`participantID`	integer	The unique ID of this participant, assigned by the TelePresence Server. **Deprecated**. Use **`participantGUID`** instead.
`conferenceGUID`	string	Globally unique identifier of the conference.
`conferenceID`	integer	Unique conference identifier. **Deprecated**. Use **`conferenceGUID`** instead.
`address`	string	The address of the participant.
`endpointCategory`	string	■ normal: Normal endpoint ■ group: Grouped endpoints ■ t3: T3 ■ cts: Cisco CTS or other TIP capable endpoints This parameter's value may not be correct in the case of participants whose calls have not yet been established at the time of enumeration.
`callProtocol`	string	■ sip: This call uses the SIP protocol. ■ h323: This call uses the H.323 protocol.
`callDirection`	string	This parameter is not present if **`callState`** is **`0`** (not connected). ■ incoming: The participant called in to the TelePresence Server ■ outgoing: The TelePresence Server called out to the participant
`groupAddressList`	array of strings	Each member of the array is an address of one of the group members. This array is only returned for endpoint groups; that is, when the **`address`** of the particpant in the conference.invite **`participants`** array was set to a comma-separated list of addresses. The index position of each endpoint's address corresponds with the position in the comma-separated list provided in the **`address`** parameter.

participant.set

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

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

Table 197: **`participant.set`** inputs

Parameter name	Type	Description
`participantGUID`	string	**Required**. The GUID of this participant, assigned by the TelePresence Server.

Table 197: participant.set inputs (continued)

Parameter name	Type	Description
participantID	integer	The unique ID of this participant, assigned by the TelePresence Server. **Deprecated**. Use **participantGUID** instead.
txAudioMute	boolean	Defines whether the TelePresence Server mutes the audio signal transmitted to this endpoint.
rxAudioMute	boolean	Defines whether the TelePresence Server mutes the audio signal received from this endpoint.
txVideoMute	boolean	Defines whether the TelePresence Server mutes the video signal transmitted to this endpoint.
rxVideoMute	boolean	Defines whether the TelePresence Server mutes the video signal received from this endpoint.
isImportant	boolean	Defines whether the participant is important (i.e. the participant's transmitted video is given preference over others when composing video). ▪ true: The participant is important ▪ false: (Default if omitted) The participant's video is not given preference over other that of the other participants
defaultLayoutSingleScreen	string	One of **single**, **activePresence**, **equal**, or **prominent**. Defines which layout should be displayed on the participant's endpoint if it is a single-screen endpoint. This parameter is ignored if the participant is using a multiscreen endpoint.
defaultLayoutMultiScreen	string	One of **single** or **activePresence**. Defines which layout should be displayed on the participant's endpoint if it is a multiscreen endpoint. This parameter is ignored if the participant is using a single-screen endpoint.
forceDefaultLayout	boolean	Defines whether the layout sent to the participant is forced to the default for their endpoint type, or whether the user may change the layout. ▪ true: The layout is forced to be the default for the user's endpoint type (the value of either **defaultLayoutSingleScreen** or **defaultLayoutMultiScreen**). The user cannot change the layout. ▪ false: The layout is not forced; the user may change the layout if the endpoint is capable.
automaticGainControl	boolean	Defines whether automatic gain control is enabled. If not specified, the conference default is used.

participant.tidylayout

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

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

Table 198: **`participant.tidylayout`** inputs

Parameter name	Type	Description
`participantGUID`	string	**Required**. The GUID of this participant, assigned by the TelePresence Server.
`participantID`	integer	The unique ID of this participant, assigned by the TelePresence Server. **Deprecated**. Use **`participantGUID`** instead.

system.info

Returns the current status of the queried system. This command takes no input parameters.

Table 199: **`system.info`** returned data

Parameter name	Type	Description
`platform`	string	The TelePresence Server's platform, as it appears in **system.xml**.
`operationMode`	string	One of **`standalone`** (locally managed), **`flexible`** (remotely managed), or **`slave`** (slave blade in a cluster).
`licenseMode`	string	Depends on the value of **`operationMode`**: Either **`HD`** or **`fullHD`**, if **`operationMode`** is **`standalone`** Always **`flexible`** if **`operationMode`** is **`flexible`** Absent if **`operationMode`** is **`slave`**
`numControlledServers`	integer	Number of TelePresence Servers controlled by this unit (including itself).
`clusterType`	string	The cluster status of this device. One of **`master`**, **`slave`**, or **`unclustered`**.
`gateKeeperOK`	boolean	Whether the gatekeeper is configured and registered.
`tpsNumberOK`	integer	Number of configured and active TelePresence Servers.
`tpdVersion`	string	TelePresence Server version number.
`tpdName`	string	TelePresence Server system name.
`tpdUptime`	integer	Period of time (in seconds) that has passed since the system booted.
`tpdSerial`	string	TelePresence Server serial number.
`makeCallsOK`	boolean	**`True`** if the system has enough resources to make at least one call.
`portsVideoTotal`	integer	The total number of video ports.
`portsVideoFree`	integer	Count of the currently unused video ports.
`portsAudioTotal`	integer	The total number of audio ports.
`portsAudioFree`	integer	Count of the currently unused audio ports.
`portsContentTotal`	integer	The total number of content ports.

Table 199: system.info returned data (continued)

Parameter name	Type	Description
`portsContentFree`	integer	Count of the currently unused content ports.
`maxConferenceSizeVideo`	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`	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`	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.
`softwareVersion`	string	Software version string eg. *4.0(1.25)*

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>
    <product>TS</product>
    <platform>8710</platform>
    <productDisplayName>Cisco TelePresence Server</productDisplayName>
    <platformDisplayName>8710</platformDisplayName>
    <serial>SM021037</serial>
    <softwareVersion>3.1(1.45)</softwareVersion>
    <buildVersion>13.1(1.45)</buildVersion>
    <hostName>A host name</hostName>
    <ipAddress>198.51.100.14</ipAddress>
    <ipAddressV6>2001:DB8::81b7</ipAddressV6>
    <macAddress>BA:98:76:54:32:10</macAddress>
    <gatekeeperUsage>Yes</gatekeeperUsage>
    <gatekeeperAddress>mainvcs.test.lal</gatekeeperAddress>
    <gatekeeperIds>dt12b7,dt12b7-l,dt12b7-c,dt12b7-r</gatekeeperIds>
    <sipRegistrarUsage>Yes</sipRegistrarUsage>
    <sipRegistrarAddress>mainvcs.test.lal</sipRegistrarAddress>
    <sipRegistrarDomain>test.lal</sipRegistrarDomain>
    <sipTrunkUsage>No</sipTrunkUsage>
    <sipTrunkAddress/>
    <sipTrunkDomain/>
    <isMaster>Yes</isMaster>
    <clusterType>unclustered</clusterType>
    <totalVideoPorts>12</totalVideoPorts>
    <totalContentPorts>12</totalContentPorts>
    <totalAudioOnlyPorts>10</totalAudioOnlyPorts>
    <uptimeSeconds>230641</uptimeSeconds>
  </system>
```

Table 200: System XML contents

Node name	Node contents
manufacturer	TANDBERG
model	Telepresence Server <model number> eg. *Telepresence Server 8710*
product	TS
platform	<platform> eg. *Media 310*, *8710*, or *Virtual Machine with 16 vCPUs*
productDisplayName	*Cisco TelePresence Server*. The display name values are subject to change with new software releases, so your application should not rely on them.

Table 200: System XML contents (continued)

Node name	Node contents
platformDisplayName	<platform> eg. *Media 310*, *8710*, or *Virtual Machine with 16 vCPUs*. The display name values are subject to change with new software releases, so your application should not rely on them.
serial	Unique serial number of the unit
softwareVersion	Software version string eg. *4.0(1.25)*
buildVersion	Build number string eg. *13.2(1.25)*
hostName	Host name of the unit
ipAddress	IPv4 address
ipAddressV6	IPv6 address
macAddress	MAC address
gatekeeperUsage	**`Yes`**: gatekeeper usage is enabled **`No`**: gatekeeper usage is disabled
gatekeeperAddress	The gatekeeper host name or IP address
gatekeeperIds	Comma separated list of registered IDs associated with this TelePresence Server and its slaves (omitted if the system is not a master)
sipRegistrarUsage	**`Yes`**: registrar usage is enabled **`No`**: registrar usage is disabled
sipRegistrarAddress	SIP registrar host name / IP address
sipRegistrarDomain	SIP registrar domain
sipTrunkUsage	**`Yes`**: trunk usage is enabled **`No`**: trunk usage is disabled
sipTrunkAddress	SIP trunk host name / IP address
sipTrunkDomain	SIP trunk domain
isMaster	**`Yes`**: this system is a master, or it is unclustered **`No`**: this system is a slave
clusterType	The role of this system in a cluster. May be **`unclustered`**, **`master`**, or **`slave`**
totalVideoPorts	Total number of video ports
totalContentPorts	Total number of video content ports
totalAudioOnlyPorts	Total number of audio-only ports
uptimeSeconds	System uptime in seconds

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.

Table 201: Fault codes

Fault Code	Description
1	`method not supported`. This method is not supported on this device or is unknown.
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.
13	`invalid PIN`. A PIN specified is not a valid series of digits.
14	`authorization failed`. The requested operation is not permitted because the supplied authentication parameters were not recognized.
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`. There are insufficient free ports to complete/place the requested calls.
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	`port limit lower than active`. New port limit is lower than currently active.
34	`internal error`. An error occurred while processing the API request.
35	`string is too long`. The call supplied a string parameter that was longer than allowed.
61	The removal of a feature or license key failed for one of several reasons. The fault code message will vary depending on the underlying cause of the failure.
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 an enumerated type contains an invalid value.
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".

Table 201: Fault codes (continued)

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.
202	**`Product needs its activation feature key`**. This request requires that the product is activated.
203	**`Too many asynchronous requests`**. The TelePresence Server is currently dealing with the maximum number of asynchronous requests of this type. Please retry this request later.
204	**`Too many invalid keys entered. Wait 5 seconds to retry`**. The TelePresence Server will not currently accept more requests to add feature keys.

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>ceb2b610-777e-11e3-b6e1-000d7c10c7d0</string>
            </value>
          </member>
          <member>
            <name>sourceIdentifier</name>
            <value>
              <string>00:0D:7C:10:C7:D0</string>
            </value>
          </member>
          <member>
            <name>audioRx</name>
            <value>
              <array>
                <data>
                  <value>
                    <struct>
                      <name>codec</name>
                      <value>
                        <string>AAC-LD</string>
                      </value>
                      <name>encrypted</name>
                      <value>
                        <boolean>1</boolean>
                      </value>
                      <name>channelBitRate</name>
                      <value>
                        <int>128000</int>
                      </value>
                      <name>jitter</name>
                      <value>
                        <int>0</int>
                      </value>
                      <name>energy</name>
                      <value>
                        <int>-25</int>
                      </value>
                      <name>packetsReceived</name>
                      <value>
                        <int>7750</int>
                      </value>
                      <name>packetErrors</name>
                      <value>
                        <int>0</int>
                      </value>
                      <name>packetsMissing</name>
                      <value>
                        <int>0</int>
                      </value>
                      <name>framesReceived</name>
                      <value>
                        <int>7749</int>
                      </value>
                      <name>frameErrors</name>
                      <value>
                        <int>0</int>
                      </value>
                      <name>muted</name>
                      <value>
                        <boolean>0</boolean>
                      </value>
                      <name>clearPathOverhead</name>
                      <value>
                        <int>0</int>
                      </value>
                      <name>clearPathRecovered</name>
                      <value>
                        <int>0</int>
                      </value>
                    </struct>
                  </value>
                </data>
              </array>
            </value>
          </member>
          <member>
            <name>audioTx</name>
            <value>
              <array>
                <data>
                  <value>
                    <struct>
                      <name>codec</name>
                      <value>
                        <string>AAC-LD</string>
                      </value>
                      <name>encrypted</name>
                      <value>
                        <boolean>1</boolean>
                      </value>
                      <name>channelBitRate</name>
                      <value>
                        <int>128000</int>
                      </value>
                      <name>packetsSent</name>
                      <value>
                        <int>7750</int>
                      </value>
                      <name>muted</name>
                      <value>
                        <boolean>0</boolean>
                      </value>
                      <name>packetsLost</name>
                      <value>
                        <int>0</int>
                      </value>
                      <name>clearPathOverhead</name>
                      <value>
                        <int>0</int>
                      </value>
                      <name>clearPathRecovered</name>
                      <value>
                        <int>0</int>
                      </value>
                    </struct>
                  </value>
                </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>720</int>
                      </value>
                      <name>width</name>
                      <value>
                        <int>1280</int>
                      </value>
                      <name>encrypted</name>
                      <value>
                        <boolean>1</boolean>
                      </value>
                      <name>channelBitRate</name>
                      <value>
                        <int>4000000</int>
                      </value>
                      <name>expectedBitRate</name>
                      <value>
                        <int>4000000</int>
                      </value>
                      <name>expectedBitRateReason</name>
                      <value>
                        <string>notLimited</string>
                      </value>
                      <name>actualBitRate</name>
                      <value>
                        <int>3758993</int>
                      </value>
                      <name>jitter</name>
                      <value>
                        <int>5</int>
                      </value>
                      <name>packetsReceived</name>
                      <value>
                        <int>61457</int>
                      </value>
                      <name>packetErrors</name>
                      <value>
                        <int>0</int>
                      </value>
                      <name>framesReceived</name>
                      <value>
                        <int>8888</int>
                      </value>
                      <name>frameErrors</name>
                      <value>
                        <int>1</int>
                      </value>
                      <name>frameRate</name>
                      <value>
                        <int>60</int>
                      </value>
                      <name>fastUpdateRequestsSent</name>
                      <value>
                        <int>3</int>
                      </value>
                      <name>muted</name>
                      <value>
                        <boolean>0</boolean>
                      </value>
                      <name>clearPathOverhead</name>
                      <value>
                        <int>0</int>
                      </value>
                      <name>clearPathRecovered</name>
                      <value>
                        <int>0</int>
                      </value>
                    </struct>
                  </value>
                </data>
              </array>
            </value>
          </member>
```

```
<member>
  <name>videoTx</name>
  <value>
    <array>
      <data>
        <value>
          <struct>
            <name>codec</name>
            <value>
              <string>H.264</string>
            </value>
            <name>height</name>
            <value>
              <int>720</int>
            </value>
            <name>width</name>
            <value>
              <int>1280</int>
            </value>
            <name>encrypted</name>
            <value>
              <boolean>1</boolean>
            </value>
            <name>channelBitRate</name>
            <value>
              <int>4000000</int>
            </value>
            <name>configuredBitRate</name>
            <value>
              <int>4000000</int>
            </value>
            <name>configuredBitRateReason</name>
            <value>
              <string>notLimited</string>
            </value>
            <name>actualBitRate</name>
            <value>
              <int>3965252</int>
            </value>
            <name>packetsSent</name>
            <value>
              <int>60599</int>
            </value>
            <name>frameRate</name>
            <value>
              <int>60</int>
            </value>
            <name>fastUpdateRequestsReceived</name>
            <value>
              <int>0</int>
            </value>
            <name>muted</name>
            <value>
              <boolean>0</boolean>
            </value>
            <name>packetsLost</name>
            <value>
              <int>0</int>
```

```
                      </value>
                      <name>clearPathOverhead</name>
                      <value>
                        <int>0</int>
                      </value>
                      <name>clearPathRecovered</name>
                      <value>
                        <int>0</int>
                      </value>
                      <name>clearPathLTRF</name>
                      <value>
                        <boolean>1</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>contentVideoRx</name>
            <value>
              <array>
                <data>
                  <value>
                    <struct>
                      <name>codec</name>
                      <value>
                        <string>H.263+</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>2000000</int>
                      </value>
                      <name>expectedBitRate</name>
                      <value>
                        <int>2000000</int>
                      </value>
                      <name>expectedBitRateReason</name>
                      <value>
                        <string>notLimited</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>
                    </struct>
                  </value>
                </data>
              </array>
            </value>
          </member>
          <member>
            <name>contentVideoTx</name>
            <value>
              <array>
                <data />
              </array>
            </value>
          </member>
```

```
        </struct>
      </value>
    </param>
  </params>
</methodCall>
```

Locally managed API change history

Table 202: API version 4.0(2.8) change summary

XML-RPC Request / Topic	Parameter	Change
callHome.configure [p.120]	**`mode`**, **`automatic`**	New command
callHome.query [p.121]	**`mode`**, **`automatic`**	New command

Table 203: API version 4.0 change summary

XML-RPC Request / Topic	Parameter	Change
device.feature.add [p.144]	**`key`**	New command
device.feature.remove [p.144]	**`key`**	New command
device.query [p.144]	**`mediaResourceRestarts`**, **`key`**, **`expiry`**	Added
device.query [p.144]	**`currentTime`**, **`restartTime`**, **`uptime`**	Documentation corrected
Fault codes [p.161]	61, 204	New fault codes
system.info [p.158]	**`softwareVersion`**	Added
conference.invite [p.128]	**`allowStarSixMuting`**	Added

Table 204: API version 3.1 change summary

XML-RPC Request / Topic	Parameter	Change
system.info [p.158]	**`clusterType`**	Added
device.query [p.144]	**`activatedLicenses`**, **`portsVideoTotal`**, **`portsAudioTotal`**	Added
conference.create [p.123]	**`automaticGainControl`**, **`encryptionRequired`**	Added
conference.invite [p.128]	**`alwaysReconnect`**, **`deferConnect`**, **`autoDisconnect`**, **`defaultLayoutSingleScreen`**, **`defaultLayoutMultiScreen`**, **`forceDefaultLayout`**, **`automaticGainControl`**, **`groupAddressList`**	Added

Table 204: API version 3.1 change summary (continued)

XML-RPC Request / Topic	Parameter	Change
conference.invite [p.128], conference.senddtmf [p.133]	**`dtmf`**	Modified
conference.set [p.134]	**`automaticGainControl`**, **`encryptionRequired`**	Added
conference.status [p.136]	**`automaticGainControl`**, **`encryptionRequired`**, **`groupAddressList`**, **`groupCallStateList`**, **`groupAudioIndex`**, **`groupContentIndex`**	Added
participant.diagnostics [p.152]	**`clearPathOverhead`**, **`clearPathRecovered`**, **`packetsLost`**, **`clearPathLTRF`**, **`clearPathLTRFRepaired`**	Added
participant.enumerate [p.155]	**`groupAddressList`**	Added
participant.set [p.156]	**`defaultLayoutSingleScreen`**, **`defaultLayoutMultiScreen`**, **`forceDefaultLayout`**, **`automaticGainControl`**	Added

Table 205: API version 3.0 change summary

XML-RPC Request	Parameter	Change
system.info	**`operationMode`** **`licenseMode`**	Modified

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.

© 2014 Cisco Systems, Inc. All rights reserved.