Cisco TelePresence IP Gateway Remote Management API 2.10

Reference Guide

D14660.04

November 2012

Contents

Introduction

This guide accompanies Version 2.10 of the Cisco TelePresence IP Gateway Remote Management API. The following Cisco TelePresence IP Gateway products support this API provided that they are running software Version 2.0(3.*x*) or later:

- IP GW 3500 Series
- IP GW MSE 8350

Terminology

For clarity this guide uses the following conventions:

Term	Meaning
API	The remote management API for the Cisco TelePresence IP Gateway products.
Device	The specific model of the Cisco TelePresence IP Gateway that you are using.
Application	The calling application that sends API commands to the device.

API version history

The following table lists the available API versions and indicates which software versions of the Cisco TelePresence IP Gateway devices support which API versions:

API version	Device software version
2.10	2.0(3.*x*) and later
2.5	2.0(1.*x*)

API implementation

XML-RPC

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 interface is stateless. Currently there is no mechanism for the device to call back the controlling application, so the controlling application must poll the device for status as required.

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>20050218T10:45:00</dateTime.iso8601></value>
</member>
```

Note: Unless otherwise stated in this guide, all strings have a maximum length of 31 characters.

For more information about XML-RPC see the XML-RPC specification.

Transport

The device implements HTTP/1.1 as defined by RFC 2616. It expects to receive HTTP communications over TCP/IP connections to port 80. The application should send HTTP POST messages to the URL defined by path **/RPC2** on the device's IP address.

The device also supports HTTPS, provided that it is running software version 2.0 or later. By default HTTPS is provided on TCP port 443. Optionally you can configure the device to receive HTTP and HTTPS connections on non-standard TCP port numbers.

Consider API overhead when writing applications

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).

It is important to bear this overhead in mind when designing your application architecture and software. The Design considerations [p.8] section provides recommendations for minimizing API overhead.

API messaging overview

Your application can send command messages (calls) to the Cisco TelePresence IP Gateway. Command messages are sent in XML format. The available command messages are detailed in the API reference section, including their parameters (required and optional) and expected responses.

Authentication

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

To manage the Cisco TelePresence IP Gateway, the controlling application must authenticate itself on the device as a user with relevant privileges. As the interface is stateless, it follows that every message must provide appropriate authentication. Depending on the certificate authentication options configured for the Cisco TelePresence IP Gateway, this can be provided in authentication parameters or by presenting a valid client certificate (see Common message elements [p.7] for details).

All calls require administrator privileges.

Message flow

The API message flow is as follows:

1. The application initiates the communication.
2. For each command sent (provided that the message is correctly formatted according to the XML-RPC specification) the device responds with a message that indicates success or failure.
   The response message may also contain any data that was requested. In the case of an error condition, the response may include only a fault code.

Examples of command and response messages are provided in API message examples [p.6]. Associated fault codes are listed in the Fault codes [p.15] section.

Encoding (ASCII or Unicode)

Your application can encode messages as ASCII text or as UTF-8 Unicode. If no method is specified, the API assumes ASCII.

If you want to specify the encoding, you can do it in a number of ways:

- Specifying encoding with HTTP headers
  Use the "Accept-Encoding: utf-8" header *or* modify the Content-Type header to read "Content-Type: text/xml; charset=utf-8".
- Specifying encoding with XML header
  The <?xml> tag is required at the top of each XML file. The API will accept an additional encoding parameter with value UTF-8 for this tag. That is, <?xml version="1.0" encoding="UTF-8"?>.

API message examples

Example command

This is an example of a command message that queries the path for the Cisco TelePresence Management Suite (Cisco TMS) address book.

```
<?xml version='1.0'?>
  <methodCall>
    <methodName>corpdirURI.query</methodName>
      <params>
        <param>
          <value><struct>
              <member>
                <name>authenticationPassword</name>
                <value><string></string></value>
              </member>
              <member>
                <name>authenticationUser</name>
                <value><string>admin</string></value>
              </member>
          </struct></value>
        </param>
      </params>
  </methodCall>
```

Example response

Assuming that the command was well formed and the device is responsive, the device will respond with an XML methodResponse message. This is an example of a response message, which returns the data requested by the previous example.

```
<?xml version="1.0"?>
  <methodResponse>
    <params>
      <param>
        <value>
          <struct>
            <member>
              <name>corpdirURI</name>
              <value>
                <string>http://tms1.cisco.com/</string>
              </value>
            </member>
          </struct>
        </value>
      </param>
    </params>
  </methodResponse>
```

Common message elements

Authentication parameters

All messages must contain a user name and password, as follows:

Parameter	Type	Comments
authenticationUser	String	The name of a user with sufficient privilege for the operation being performed. The name is case sensitive.
authenticationPassword	String	The password that corresponds with the given user. The API ignores this parameter if the user has no password (this behavior differs from the web interface, where the password entry field must be blank).

Note: All calls require administrator privileges.

Design considerations

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.

API connection limits

The current API implementation accepts a maximum of four concurrent XML-RPC requests and supports a maximum of eight concurrent TCP connections.

Using HTTP keep-alives

If you are using API version 2.4 or later, your application can use HTTP keep-alives to reduce the amount of TCP traffic that results from constantly polling the device. For details, see HTTP keep-alives [p.9].

HTTP keep-alives

Note: This feature is available with API version 2.4 and later.

Your application can use HTTP keep-alives to reduce the amount of TCP traffic that results from constantly polling the device.

Implementation

Any client that supports HTTP keep-alives may include the following line in the HTTP header of an API request:

```
Connection: Keep-Alive
```

This line indicates to the device that the client supports HTTP keep-alives. The device *may* then decide that it will maintain the TCP connection after it has responded to the request.

If the device decides that it will not maintain the connection after it has responded, the device returns the following line in the HTTP header of its response:

```
Connection: close
```

Subject to the limitations mentioned below, if this line is absent from the HTTP header of the response, the device will keep the TCP connection open and the client may use the same connection for a subsequent request.

Limitations

The device will not allow a connection to be kept alive if:

- The current connection has already serviced a set number of requests.
- The current connection has already been open for a set amount of time.
- There are already more than a certain number of connections in a kept-alive state.

These restrictions are in place to limit the resources associated with kept-alive connections. If a connection is terminated for either of the first two reasons, the client will probably find that the connection is maintained (back in a keep-alive state) after the next request.

Usage considerations

The client should never assume that a connection will be maintained.

Also, even after a response that does not contain the `Connection:close` header, the device will close an open connection if no further requests are made by the client within one minute. If requests from the client are likely to be this far apart, there is little to be gained by using HTTP keep-alives.

API reference

The Cisco TelePresence IP Gateway supports the following API calls:

This reference section provides a description of each call, including (where applicable):

- Call function
- Accepted parameters
- Returned parameters, structure formats, and data types
- Deprecated parameters
- Additional information

cdrlog.delete

Parameter	Type	Comments
deleteIndex	Integer	You can delete logs in blocks of 400 entries. To delete logs, use either of these methods: ■ Enter the deleteableIndex value returned after a log query call. This deletes all complete blocks (400 entries) below this value, and leaves any residual entries. ■ To limit the delete operation to fewer entries, enter an appropriate number that is lower than the value of deleteableIndex. This deletes all complete blocks below the specified number, and leaves any residual entries.

Stored events up to and including the indicated deleteIndex value will be permanently deleted.

cdrlog.query

Deletes stored events from the CDR log.

This call takes no parameters. The response returns the following:

Response	Type	Comments
firstIndex	Integer	The index of the oldest stored event.
deleteableIndex	Integer	The index of the most recent deletable event.
numEvents	Integer	The total number of events stored.
percentageCapacity	Integer	The percentage of total available capacity used by the log.

corpdirURI.configure

This method call configures the path to the Cisco TelePresence Management Suite (Cisco TMS) address book.

CAUTION: This call should *not* be called by users. Any user changes are likely to break functionality and to be overwritten automatically.

Parameter	Type	Comments
corpdirURI	String (length <256)	The full path of the Cisco TMS address book.

corpdirURI.query

This method call takes no parameters. The method response returns the following:

Parameter	Type	Comments
corpdirURI	String (length <256)	The full path of the Cisco TMS address book.

device.health.query

This call takes no parameters. The response returns the current status of the device, such as health monitors and CPU load.

Note: Some values do not apply to all device types.

Response	Type	Comments
cpuLoad	Integer	The CPU load, as a percentage value.
mediaLoad	Integer	The media processor loads, as percentage values. The load values are returned as a total and split between audio and video.
audioLoad	Integer	
videoLoad	Integer	
fanStatus	String	One of ok, outOfSpec or critical.
fanStatusWorst	String	
temperatureStatus	String	
temperatureStatusWorst	String	
rtcBatteryStatus	String	
rtcBatteryStatusWorst	String	
voltagesStatus	String	
voltagesStatusWorst	String	
operationalStatus	String	One of active, shuttingDown or shutDown.

device.network.query

This call takes no parameters. The response returns the following XML-RPC structures:

Response	Type	Comments
portA	Struct (see following table)	Contains the configuration and status for port A.
portB	Struct (see following table)	Contains the configuration and status for port B.

The format for the portA and portB structs is as follows:

Field	Type	Comments
enabled	Boolean	True if the port is enabled, false otherwise.
linkStatus	Boolean	True if the link is up, false if the link is down.
speed	Integer	One of 10, 100, or 1000 (in Mb/s).
fullDuplex	Boolean	True if full duplex enabled, false if half.
macAddress	String	A 12-character string; no separators.
packetsSent	Integer	Statistics from the web interface (these values are 32-bit signed integers, so may wrap).
packetsReceived	Integer	
multicastPacketsSent	Integer	
multicastPacketsReceived	Integer	
bytesSent	Integer	
bytesReceived	Integer	
queueDrops	Integer	
collisions	Integer	
transmitErrors	Integer	
receiveErrors	Integer	
bytesSent64	String	64-bit versions of the above statistics, using a string rather than an integer.
bytesReceived64	String	
Optional fields		
hostName	String	Host name of the device.
dhcp	Boolean	True if IP address is configured by DHCP, false otherwise.
ipAddress	String	IP address.
subnetMask	String	IP subnet mask.
defaultGateway	String	Default gateway IP address.
domainName	String	Domain name of the device.
nameServer	String	IP address.
nameServerSecondary	String	IP address.

Optional fields are returned only if the interface is both enabled and configured.

device.query

This method call takes no parameters. The method response returns the following:

Response	Type	Comments
currentTime	dateTime.iso8601	The current system time (UTC format).
restartTime	dateTime.iso8601	The date and time at which the device was last booted.
serial	String	The serial number of the device.
softwareVersion	String	The software version of the running software.
buildVersion	String	The build version of the running software.
model	String	The model type of this device.
apiVersion	String	The version number of the API implemented by this device.
activatedFeatures	Array	Currently only contains a string "feature" with a short description of the feature.
totalVideoPorts	Integer	The number of video ports on the device.
totalAudioOnlyPorts	Integer	The number of additional audio-only ports on the device.
maxVideoResolution	String	One of cif or 4cif.

device.restartlog.query

This call takes no parameters. The response returns the restart log (in the application web interface the restart log is also known as the system log).

Response	Type	Comments
log	Array	Contains the restart log in structures as described below.

The format for the log array is as follows:

Field	Type	Comments
time	dateTime.iso8601	The time of the last reboot.
reason	String	The reason for the reboot (one of unknown, User requested shutdown, or User requested upgrade).

system.xml information

You can derive some information about the device from its system.xml file, which you can download via HTTP from the device root (for example, **http://TestIPGW/system.xml**). Information available in the system.xml file includes the manufacturer, model type, and serial number of the device.

Example system.xml

```
<?xml version="1.0"?>
  <system>
    <manufacturer>Cisco/manufacturer>
    <model>IP GW 3520</model>
    <serial>MRV1001SM0002D9</serial>
    <softwareVersion>2.0(1.7)</softwareVersion>
    <buildVersion>5.3(1.7)</buildVersion>
    <hostName>TestIPGW</hostName>
    <uptimeSeconds>2345</uptimeSeconds>
  </system>
```

Fields in the system.xml file

Field	Comments
manufacturer	The manufacturer of the device.
model	The model type of the device.
serial	The serial number of the device.
softwareVersion	The software version that is currently running.
buildVersion	The build version of the software that is currently running.
hostName	The host name of the system.
uptimeSeconds	The number of seconds since boot.

Fault codes

In common with certain other Cisco TelePresence applications, the Cisco TelePresence IP Gateway returns a fault code when a fault occurs during processing of an XML-RPC request.

The individual call descriptions in this guide give some indication of which faults may occur. The following table describes all possible fault codes that are used within this specification and their most common interpretations. Not all codes are used by the Cisco TelePresence IP Gateway.

Fault code	Description
1	**Method not supported**. This method is not supported on this device.
2	**Duplicate conference name**. A conference name was specified, but is already in use.
3	**Duplicate participant name**. A participant 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.
7	**Too many participants**. Too many participants are already configured and no more can be created.
8	**No conference name or auto attendant id supplied**. A conference name or auto attendant identifier was required, but was not present.
9	**No participant name supplied**. A participant name is required but was not present.
10	**No participant address supplied**. A participant address is required but was not present.
11	**Invalid start time specified**. A conference start time is not valid.
12	**Invalid end time specified**. A conference end time is not valid.
13	**Invalid PIN specified**. A specified PIN is not a valid series of digits.
14	**Authorization failed**. This code may be returned for a failed login attempt, in which case the supplied username or password, or both, may be incorrect.
15	**Insufficient privileges**. The specified user id and password combination is not valid for the attempted operation.
16	**Invalid enumerateID value**. An enumerate ID passed to an enumerate method invocation was invalid. Only values returned by the device should be used in enumerate methods.
17	**Port reservation failure**. This is in the case that reservedAudioPorts or reservedVideoPorts value is set too high, and the device cannot support this.
18	**Duplicate numeric ID**. A numeric ID was given, but this ID is already in use.
19	**Unsupported protocol**. A protocol was used which does not correspond to any valid protocol for this method. In particular, this is used for participant identification where an invalid protocol is specified.
20	**Unsupported participant type**. A participant type was used which does not correspond to any participant type known to the device.
21	**No such folder**. A folder identifier was present, but does not refer to a valid folder.
22	**No such recording**. A recording identifier was present, but does not refer to a valid recording.

Fault code	Description
23	**No changes requested**. This is given when a method for changing something correctly identifies an object, but no changes to that object are specified.
24	**No such port**. This is returned when an ISDN or serial port is given as a parameter which does not exist on an ISDN or serial gateway device.
101	**Missing parameter**. This is given when a required parameter is absent. The parameter in question is given in the fault string in the format "missing parameter - *parameter_name*".
102	**Invalid parameter**. This is given when a parameter was successfully parsed, is of the correct type, but falls outside the valid values. For example, an integer is too high or a string value for a protocol contains an invalid protocol. The parameter in question is given in the fault string in the format "invalid parameter - *parameter_name*".
103	**Malformed parameter**. This is given when a parameter of the correct name is present, but cannot be read for some reason. For example, the parameter is supposed to be an integer but is given as a string. The parameter in question is given in the fault string in the format "malformed parameter - *parameter_name*".
201	**Operation failed**. This is a generic fault for when an operation does not succeed as required.

Bibliography

All documentation for the latest versions of the Cisco TelePresence products covered in this guide can be found on Cisco.com. Other documents referred to in this guide can be found at:

Title	Reference	Link
RFC 2616: Hypertext Transfer Protocol - HTTP/1.1	RFC 2616	http://www.faqs.org/rfcs
XML-RPC specification		www.xmlrpc.com