



# Cisco TelePresence IP Gateway Remote Management API Reference Guide

---

D14660.03

December 2010

---

# Contents

<b>API history</b> .....	<b>3</b>
<b>Introduction</b> .....	<b>4</b>
HTTP and HTTPS .....	4
Consider API overhead when writing applications.....	4
<b>Protocol overview</b> .....	<b>5</b>
Authentication .....	5
Message flow .....	5
Unicode support .....	10
HTTP headers.....	10
XML header.....	10
Common message elements .....	11
Authentication .....	11
<b>API reference</b> .....	<b>12</b>
corpdirURI.configure .....	13
corpdirURI.query .....	13
device.health.query .....	14
device.network.query .....	15
device.query .....	17
device.restartlog.query.....	18
<b>Related information sources</b> .....	<b>19</b>
system.xml .....	19
<b>Fault codes</b> .....	<b>20</b>
<b>HTTP keep-alives</b> .....	<b>22</b>
<b>References</b> .....	<b>23</b>

## API history

The latest version of the Cisco TelePresence IP Gateway Remote Management API is version 2.5. The following Cisco TelePresence IP Gateway products support this API version when running software version 2.0, and later:

- ▶ IP GW 3500 Series
- ▶ IP GW MSE 8350 blade

The following table shows which version of Cisco TelePresence products support this version.

API Version	IP GW 3500 Series, IP GW MSE 8350
2.4	N/A
2.5	2.0, and later

# Introduction

This reference guide contains the specification of the Cisco TelePresence IP Gateway Remote Management API, by which it is possible to control the following Cisco TelePresence products:

- ▶ IP GW 3500 Series
- ▶ IP GW MSE 8350 blade

This is accomplished via messages sent using the XML-RPC protocol. XML-RPC is a simple protocol for remote procedure calling using HTTP as the transport and XML as the encoding. It is designed to be as simple as possible, whilst allowing complex data structures to be transmitted, processed and returned. XML-RPC has no platform or software dependence and was chosen over SOAP because of its simplicity.

The interface is stateless. Currently, there is no mechanism for the Cisco TelePresence IP Gateway to call back the controlling application and therefore the controlling application must poll the IP gateway for status, as required. A future enhancement *may* provide a mechanism for signaling device status changes to the controlling application.

In this implementation of XML-RPC all parameters and return values are part of a <struct> and are explicitly named. For example, the device.query call returns the current time value as a structure member named 'currentTime' rather than as a single value of type <dateTime.iso8601>.

---

**Note:** Unless otherwise stated, assume string length is a maximum of 32 characters.

---

For further details of XML-RPC refer to the [specification](#) [1].

## HTTP and HTTPS

Cisco TelePresence IP Gateways expect to receive HTTP communication over TCP/IP connections to port 80. The HTTP messages should be "POST"s to the URL "/RPC2".

HTTPS (a secure, encrypted version of HTTP) is supported on all Cisco TelePresence IP Gateway products from software version 2.0 and later.

By default, HTTPS is provided on TCP port 443, although Cisco TelePresence IP Gateways can be configured to receive HTTP and HTTPS connections on non-standard TCP port numbers if required.

The Cisco TelePresence IP Gateway implements HTTP/1.1 as defined by RFC 2616 [2].

## Consider API overhead when writing applications

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

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

# Protocol overview

## Authentication

In order to manage the Cisco TelePresence IP Gateway, the controlling application must authenticate itself as a user with relevant privileges. Accordingly, each message contains a user name and password; see the section *Common message elements* below for details of the format. Note that authentication information is sent using plain text and should only be sent over a trusted network.

---

**Note:** All calls require administrator privileges.

---

## Message flow

An application can send command messages to the IP gateway. For each command sent (provided the message is correctly formatted according to the [XML-RPC spec](#)), the IP gateway responds with a message indicating success or failure. The response message may also contain any data that was requested.

Command messages are sent in XML format. For example, the following message queries network information on an IP gateway:

```
POST /RPC2 HTTP/1.0
Host: 10.2.134.175
User-Agent: xmlrpc.py/1.0.1 (by www.pythonware.com)
Content-Type: text/xml
Content-Length: 359

<?xml version='1.0' encoding='UTF-8'?>
<methodCall>
<methodName>device.network.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>
```

If the command was successful, the IP gateway sends a success response:

```
HTTP/1.1 200 OK
Connection: close
Content-Type: text/xml
Content-Encoding: utf8
Content-Length: 3968
```

```
<?xml version="1.0"?>
<methodResponse>
<params>
<param>
<value>
<struct>
<member>
<name>portA</name>
<value>
<struct>
<member>
<name>enabled</name>
<value>
<boolean>1</boolean>
</value>
</member>
<member>
<name>ipAddress</name>
<value>
<string>10.2.134.175</string>
</value>
</member>
<member>
<name>subnetMask</name>
<value>
<string>255.255.252.0</string>
</value>
</member>
<member>
<name>defaultGateway</name>
<value>
<string>10.2.132.1</string>
</value>
</member>
<member>
<name>nameServer</name>
<value>
<string>10.2.1.14</string>
</value>
</member>
<member>
<name>nameServerSecondary</name>
<value>
<string>0.0.0.0</string>
</value>
</member>
<member>
<name>dhcp</name>
<value>
<boolean>1</boolean>
</value>
</member>
<member>
<name>hostName</name>
<value>
<string/>
</value>
```

```
</member>
<member>
<name>domainName</name>
<value>
<string>convidia.concordant.co.uk</string>
</value>
</member>
<member>
<name>packetsSent</name>
<value>
<int>134761</int>
</value>
</member>
<member>
<name>packetsReceived</name>
<value>
<int>4974003</int>
</value>
</member>
  <member>
<name>bytesSent</name>
<value>
<int>45274297</int>
</value>
</member>
  <member>
<name>bytesReceived</name>
<name>multicastPacketsSent</name>
<member>
<name>multicastPacketsReceived</name>
<value>
<int>4795316</int>
</value>
</member>
  <member>
<name>queueDrops</name>
<value>
<int>0</int>
</value>
</member>
<member>
<name>receiveErrors</name>
<value>
<int>0</int>
</value>
</member>
  <member>
<name>collisions</name>
<value>
<int>0</int>
</value>
</member>
  <member>
<name>transmitErrors</name>
<value>
<int>0</int>
</value>
```

```
</member>
<member>
<name>speed</name>
<value>
<int>1000</int>
</value>
</member>
<member>
<name>fullDuplex</name>
<value>
<boolean>1</boolean>
</value>
<member>
<name>linkStatus</name>
<value>
<boolean>1</boolean>
</value>
</member>
<member>
<name>bytesSent64</name>
<value>
<string>45274297</string>
</value>
</member>
<member>
<name>bytesReceived64</name>
<value>
<string>493179785</string>
</value>
</member>
<member>
<name>macAddress</name>
<value>
<string>000D7C0005B2</string>
</value>
</member>
</struct>
</value>
</member>
<member>
<name>portB</name>
<value>
<struct>
<member>
<name>enabled</name>
<value>
<boolean>0</boolean>
</value>
</member>
<member>
<name>packetsSent</name>
<value>
<int>0</int>
</value>
</member>
<member>
<name>packetsReceived</name>
<value>
```

```
<int>0</int>
</value>
</member>
<member>
<name>bytesSent</name>
<value>
<int>0</int>
</value>
</member>
<member>
<name>bytesReceived</name>
<value>
<int>0</int>
</value>
</member>
<member>
<name>multicastPacketsSent</name>
<value>
<int>0</int>
</value>
</member>
<member>
<name>multicastPacketsReceived</name>
<value>
<int>0</int>
</value>
</member>
<member>
<name>queueDrops</name>
<value>
<int>0</int>
</value>
</member>
<member>
<name>receiveErrors</name>
<value>
<int>0</int>
</value>
</member>
<member>
<name>collisions</name>
<value>
<int>0</int>
</value>
</member>
<member>
<name>transmitErrors</name>
<value>
<int>0</int>
</value> </member>
<member>
<name>speed</name>
<value>
<int>1000</int>
</value>
</member>
<member>
<name>fullDuplex</name>
```

```

<value>
<boolean>1</boolean>
</value>
</member>
<member>
<name>linkStatus</name>
<value>
<boolean>1</boolean>
</value>
</member>
<member>
<name>bytesSent64</name>
<value>
<string>0</string>
</value>
</member>
<member>
<name>bytesReceived64</name>
<value>
<string>0</string>
</value>
</member>
<member>
<name>macAddress</name>
<value>
<string>000D7C0005B3</string>
</value>
</member>
</struct>
</value>
</member>
</struct>
</value>
</param>
</params>
</methodResponse>

```

The complete list of command messages, their required and optional parameters, and the expected responses are detailed in the sections below. The possible [fault codes](#) are listed later in this document.

## Unicode support

Parameters in this version of the API can be in ASCII text or unicode (UTF-8). In order to distinguish between these encodings, any of several methods can be used. If no method is present, ASCII is assumed.

## HTTP headers

There are two different ways of specifying unicode in the HTTP headers; either using "Accept-Encoding: utf-8", or modifying the Content-Type header to read "Content-Type: text/xml; charset=utf-8".

## XML header

The `<?xml>` tag is required at the top of each XML file. This API will accept an additional encoding parameter with value UTF-8 for this tag, i.e. `<?xml version="1.0" encoding="UTF-8"?>`.

---

## Common message elements

### Authentication

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

Parameter	Type	Comments
authenticationUser	String	Name of a user with sufficient privilege for the operation being performed. The name is case sensitive.
authenticationPassword	String	The corresponding user's password. This parameter is ignored if the user has no password set. Note that this differs from the web interface where a blank password must be blank.

---

**Note:** All calls require administrator privileges.

---

## API reference

This section is a reference to each of the IP gateway API calls. For each API call, the following information is provided where applicable:

- ▶ Description of the call's function
- ▶ Parameters
- ▶ List of responses
- ▶ Response data types
- ▶ Structure formats
- ▶ Additional information that will assist in using the API call
- ▶ Parameters deprecated from previous versions

The table below lists all currently supported IP gateway API calls. Click on the call name to go to the page containing a complete description of the call.

- ▶ [corpdirURI.configure](#)
- ▶ [corpdirURI.query](#)
- ▶ [device.health.query](#)
- ▶ [device.network.query](#)
- ▶ [device.query](#)
- ▶ [device.restartlog.query](#)

## corpdirURI.configure

This method call configures the path to the TMS address book. This call should **not** be called by users: any user changes will probably break functionality and be overwritten automatically.

Parameter	Type	Comments
corpdirURI	String	The full path of the TMS address book.

## corpdirURI.query

There are no parameters passed with this method call. The method response returns the following:

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

## device.health.query

Returns the current status of the IP gateway, such as health monitors and CPU load.

Response	Type	Comments
cpuLoad	Integer	The CPU load, as a percentage.
mediaLoad	Integer	Loads for the media processors (total, and split between audio and video) as percentage values.
audioLoad	Integer	
videoLoad	Integer	
temperatureStatus	String	One of ok, outOfSpec or critical.
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:

Parameter	Type	Comments
portA	Array (see below)	Contains the configuration and status for port A.
portB	Array (see below)	Contains the configuration and status for port B.

The format for the two structures above is:

Field	Type	Comments
enabled	Boolean	true if the port is enabled, otherwise false.
linkStatus	Boolean	true if the link is up, false if the link is down.
Speed	Integer	One of 10, 100 or 1000, in Mbps.
fullDuplex	Boolean	true if full duplex enabled, false if half.
macAddress	String	A 12 character string, no separators.
packetsSent	Integer	Stats from the web interface. It is worth noting that all these values are 32 bit signed integers, and thus may wrap.
packetsReceived	Integer	
multicastPacketsSent	Integer	
multicastPacketsReceived	Integer	
bytesSent	Integer	
bytesReceived	Integer	
queueDrops	Integer	
collisions	Integer	
transmitErrors	Integer	
receiveErrors	Integer	

Field	Type	Comments
bytesSent64	String	64 bit versions of the above stats, using a string rather than an integer.
bytesReceived64	String	
<b>Optional parameters</b>		
hostName	String	The host name of this port.
dhcp	Boolean	true if configured by DHCP, otherwise <b>false</b> .
ipAddress	String	a.b.c.d format.
subnetMask	String	a.b.c.d format.
defaultGateway	String	a.b.c.d format.
domainName	String	The domain name of this port.
nameServer	String	a.b.c.d format.
nameServerSecondary	String	a.b.c.d format.

All fields above marked optional will be returned only if the interface is enabled and has been configured.

## device.query

There are no parameters passed with this method call. The method response returns the following:

Parameter	Type	Comments
currentTime	dateTime.iso8601	The system's current time (UTC).
restartTime	dateTime.iso8601	The date and time at which the system was last restarted.
serial	String	The serial number of the IP gateway.
softwareVersion	String	The software version of the running software.
buildVersion	String	The build version of the running software.
model	String	The model of this IP gateway, e.g. " Cisco TelePresence IP GW 8350".
apiVersion	String	The version number of the API implemented by this IP gateway.
activatedFeatures	Array	Currently, only contains a string "feature" with a short description of the feature.
totalVideoPorts	Integer	The total number of video ports on the IP gateway.
totalAudioOnlyPorts	Integer	The total number of additional audio-only ports on the IP gateway.
maxVideoResolution	String	One of <b>cif</b> or <b>4cif</b> . Only present on MCU and VCR products.

## device.restartlog.query

Returns the restart log - also known as the system log on the web interface.

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

The "Log" array consists of structures which contain the following fields.

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

## Related information sources

### system.xml

While not strictly part of the XML-RPC API, some information can be retrieved from the system.xml file. This can be downloaded via HTTP as the file system.xml in the root of the unit, for example <http://TestIPGW/system.xml>

An example is:

```
<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>
```

The meaning of the fields is:

Field	Comments
manufacturer	The manufacturer of this IP gateway, e.g. Cisco
model	The model of this particular IP gateway, e.g. IP GW 8350.
serial	The serial number of this IP gateway.
softwareVersion	The software version currently running.
buildVersion	The build version of the software currently running.
hostName	The host name of the system.
uptimeSeconds	The number of seconds since boot.

## Fault codes

The Cisco TelePresence gateways, MCUs and VCRs have a series of fault codes which are returned when a fault occurs during the processing of an XML-RPC request. While individual call descriptions above give some indication of which faults may occur, below is a description of all possible fault codes used within this specification and the most common interpretation. Note that not all codes are used by the Cisco TelePresence IP Gateway.

Fault Code	Description
1	<b>Method not supported.</b> This method is not supported on this device.
2	<b>Duplicate conference name.</b> A conference name was specified, but is already in use.
3	<b>Duplicate participant name.</b> A participant name was specified, but is already in use.
4	<b>No such conference or auto attendant.</b> The conference or auto attendant identification given does not match any conference or auto attendant.
5	<b>No such participant.</b> The participant identification given does not match any participants.
6	<b>Too many conferences.</b> The device has reached the limit of the number of conferences that can be configured.
7	<b>Too many participants.</b> There are already too many participants configured and no more can be created.
8	<b>No conference name or auto attendant id supplied.</b> A conference name or auto attendant identifier was required, but was not present.
9	<b>No participant name supplied.</b> A participant name is required but was not present.
10	<b>No participant address supplied.</b> A participant address is required but was not present.
11	<b>Invalid start time specified.</b> A conference start time is not valid.
12	<b>Invalid end time specified.</b> A conference end time is not valid.
13	<b>Invalid PIN specified.</b> A PIN specified is not a valid series of digits.
14	<b>Unauthorised.</b> The requested operation is not permitted on this device.
15	<b>Insufficient privileges.</b> The specified user id and password combination is not valid for the attempted operation.
16	<b>Invalid enumerateID value.</b> An enumerate ID passed to an enumerate method invocation was invalid. Only values returned by the device should be used in enumerate methods.
17	<b>Port reservation failure.</b> This is in the case that reservedAudioPorts or

Fault Code	Description
	reservedVideoPorts value is set too high, and the device cannot support this.
18	<b>Duplicate numeric ID.</b> A numeric ID was given, but this ID is already in use.
19	<b>Unsupported protocol.</b> 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	<b>Unsupported participant type.</b> A participant type was used which does not correspond to any participant type known to the device.
21	<b>No such folder.</b> A folder identifier was present, but does not refer to a valid folder.
22	<b>No such recording.</b> A recording identifier was present, but does not refer to a valid recording.
23	<b>No changes requested.</b> This is given when a method for changing something correctly identifies an object, but no changes to that object are specified.
24	<b>No such port.</b> This is returned when an ISDN port is given as a parameter which does not exist on an ISDN gateway.
101	<b>Missing parameter.</b> This is given when a required parameter is absent. The parameter in question is given in the fault string in the format "missing parameter - <i>parameter_name</i> ".
102	<b>Invalid parameter.</b> 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 - <i>parameter_name</i> ".
103	<b>Malformed parameter.</b> 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 - <i>parameter_name</i> ".
201	<b>Operation failed.</b> This is a generic fault for when an operation does not succeed as required.

## HTTP keep-alives

**Note:** This feature is available from API version 2.4 onwards.

A method of reducing the amount of TCP traffic when polling the IP gateway via the API is to use HTTP keep-alives. (This method can be used with other Cisco TelePresence products that also support the API, such as the MCU, VCR and ISDN gateway.)

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 Cisco product that the client supports HTTP keep-alives. The IP gateway *may* then choose to not close the TCP connection after returning its response to the request. If the connection will be closed, the IP gateway returns the following line in the HTTP header of its response:

```
Connection: close
```

The absence of this line indicates that the IP gateway will keep the TCP connection open and that the client may use the same connection for a subsequent request.

The IP gateway 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 certain 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 back in a keep-alive state following the next request.

The client should never assume a connection will be kept alive.

Also note that, even after a response not containing the “connection: close” header, the connection will still be closed if no further requests are made within one minute. If requests from the client are likely to be this far apart then there is little to be gained by using HTTP keep-alives.

## References

The following table lists documents and web sites referenced in this document. All product documentation can be found on our [web site](#).

[1]	XML-RPC, <a href="http://www.xmlrpc.com/">http://www.xmlrpc.com/</a>
[2]	RFC 2616, <a href="http://www.faqs.org/rfcs/rfc2616.html">http://www.faqs.org/rfcs/rfc2616.html</a>

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](http://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.

© December 2010 Cisco Systems, Inc. All rights reserved.