Cisco Meeting Server

Cisco Meeting Server Release 3.6

Events Guide

23 August 2022

Contents

Change History

Date	Change Summary
August 23, 2022	New version for 3.6.
April 20, 2022	New version for 3.5.
August, 24, 2021	New version for 3.3.
April 09, 2021	New version for 3.2.

1 Introduction

Meeting Server can notify an " events client" in real-time of changes that are occurring on the Meeting Server. The Meeting Server acts as a server for the events, and the events client could be for example, a web-based management application. Cisco Meeting Management acts as an events client.

Note: You can construct your own events client, which is similar to constructing an API client. The events client needs to support HTTP and WebSocket libraries, both are available in common scripting languages like Python. The events port on the Meeting Server is the same port as you configured for the Web Admin, typically TCP port 443 on interface A.

Rather than continually poll an API resource on the Meeting Server, an events client can subscribe to an event resource to receive updates. For example, after establishing a WebSocket connection between the events client and the Meeting Server, the events client can subscribe to the event resource **`callRoster`** and receive updates on the participant list of an active conference to find out when a new participant joins, or an existing participant changes layout etc.

The Meeting Server supports three subscribable event resources:

- **`callInfo`** provides information about a specific conference,
- **`callRoster`** provides information about each participant in a conference,
- **`calls`** provides information on active conferences,

each subscribable event resource has subscribable elements, see Section 2.

1.1 Subscription overview

1.1.1 Subscription overview

When an events client makes a subscription to the Meeting Server, the subscription lists the set of resources the events client wants to subscribe to. This should be the complete list of resources that it wants to have an active subscription to, so if the events client needs to add a new resource to an existing set of subscriptions then it must include all the existing subscribed-to resources in the new request. To stop all active subscriptions, the events client should supply an empty request (this will also happen when the WebSocket connection is torn down).

Each resource being subscribed to within the subscription request is given a unique number by the events client, so that when the Meeting Server provides updates the client knows which subscription is being referred to. If the events client subscribes to the same resource as before, but with a different numeric identifier, this is treated as a new subscription by the Meeting Server, and the old subscription request is effectively torn down.

When the Meeting Server receives a new subscription request, it first replies with a simple " ack" saying it has received that subscription request and is processing it. A positive acknowledgement at this stage only means that the Meeting Server has received the request - there will be one or more follow ups giving an updated status of each element of the subscription request, and thereafter actual updates for that resource.

As an example, if the events client requests to subscribe to information on active conferences and the participant list for one specific conference, it will first get back an " ack" , and then a " subscriptionUpdate" message telling it that both subscriptions are " pending, this means that the Meeting Server is still in the process of setting up the subscriptions. A little while later the events client might get an update that says the subscription to the list of active conferences is " active" and the participant list subscription is still " pending" . The Meeting Server will also start providing updates for the active conferences list at this stage; no actual updates relating to a subscribed-to resource occur before the subscriber has been told that subscription is " active" . If the conference, whose participant list the events client subscribed to, exists (i.e. the supplied GUID still corresponds to an active conference) the events client will receive a subscription update indicating that both the active conference subscription and the participant list subscription are " active" , the events client will also start receiving participant list updates at this point. If the conference GUID for the participant list subscription isn't successfully resolved to an active conference, the subscription status will be " deactivated" . This is also the state that the subscription will change to in a subsequent " subscriptionUpdate" message when the conference ends or the events client unsubscribes from it.

For more information on the flow of messages when subscribing to event resources, see Section 5.

2 Event resources and subscribable elements

2.0.1 Event resources and subscribable elements

From version 2.4, the Meeting Server supports using events to provide real-time information to an " events client" on the following event resources:

- **`callInfo`** provides information about a specific conference,
- **`callRoster`** provides information about each participant in a conference,
- **`calls`** provides information on active conferences.

2.1 callInfo

Table 1: Information about a specific conference (call) available using subscribable event resource `callInfo`

Name	Value	Description
Request parameters		
call	ID	The id of the conference to receive updated information on.
Subscribable elements		
name	String	The name of the conference.
participants	Numeric	The number of participants currently in the conference.
distributedInstances	Numeric	The number of distributed instances of this conference that exist across the Call Bridge cluster (0 for an unclustered conference).
recording	active| inactive	One of: *active* - this conference is currently being recorded *inactive* - this conference is currently not being recorded.
endpointRecording	active| inactive	One of: *active* - this conference is currently being externally recorded by an endpoint (Lync client) *inactive* - this conference is currently not being externally recorded by an endpoint (Lync client).

Name	Value	Description
streaming	active| inactive	One of: *active* - this conference is currently being streamed *inactive* - this conference is currently not being streamed.
lockState	locked| unlocked	One of: *locked* - this conferenc. is currently locked *unlocked*- this conference is currently unlocked .
callType	coSpace| adHoc| lyncConferencing| forwarding	One of: *coSpace* - this call is a coSpace instantiation *adHoc* - this is an ad hoc multi-party call *lyncConferencing* - this call is a Meeting Server connection to a Lync-hosted conference *forwarding* - this is a forwarded or " gateway" call.
callCorrelator	ID	This value can be used to identify call legs which may be distributed across multiple Call Bridges, but which are all in the same call either in the same coSpace or an ad hoc call. Note: For calls within a coSpace, the callCorrelator value will be the same for the life time of the coSpace. For every ad hoc call, the value will be dynamically generated.
joinAudioMuteOverride	true| false	One of: *true* - new participants will be muted when joining the conference *false* - new participants will not be muted when joining the conference. This is the default if not set.

2.2 callRoster

Table 2: Information about each participant in a conference (call) available using subscribable event resource `callRoster`

Name	Value	Description
Request parameters		
call	ID	The id of the conference to receive participant updates from.
Subscribable elements		
name	String	The participant display name.
uri	URI user part	The URI associated with this participant.
state	initial| ringing| connected| onHold	Reflects the current signaling state of this participant.
direction	incoming| outgoing	One of: *incoming* – this participant dialled into the conference (the remote SIP device initiated the connection to the Meeting Server) *outgoing* – this participant was dialled out to inorder to join the conference (the call leg was established from the Meeting Server to the remote SIP device).
audioMuted	true| false	One of: *true* – the Meeting Server has muted this participant's audio *false* – the Meeting Server has not muted this participant's audio.
videoMuted	true| false	One of: *true* – the Meeting Server has muted this participant's video *false* – the Meeting Server has not muted this participant's video.
importance	Numeric	Value for this participant's importance. NULL if importance is not set or changes to unset.

Name	Value	Description
layout	allEqual| speakerOnly| telepresence| stacked| allEqualQuarters| allEqualNinths| allEqualSixteenths| allEqualTwentyFifths| onePlusFive| onePlusSeven| onePlusNine| automatic| onePlusN	The layout currently being used by this participant.
activeSpeaker	true| false	One of: *true* – this participant is currently considered an active speaker in this conference *false* – this participant is currently not considered an active speaker in this conference
presenter	true| false	One of: *true* – this participant is currently presenting (sharing their screen) in this conference *false* -this participant is currently not presenting in this conference.
endpointRecording	active| inactive	One of: *active* – this participant is currently recording the conference *inactive* – this participant is currently not recording the conference.
canMove	true|false	Indicates whether this participant can be moved using the movedParticipant API command. (From version 2.6)
canMoveToLobby	true|false	Indicates whether or not this participant can be moved to lobby. (From version 3.5)
movedParticipant	ID	If this participant was created as part of a participant move, the ID is the GUID of the participant that this participant was moved from. (From version 2.6)
movedParticipantCallBridge	ID	If this participant was created as part of a participant move, the ID is the GUID of the Call Bridge hosting the conference that this participantwas moved from. (From version 2.6)

2.3 calls

Table 3: Information on active conferences (calls) available using subscribable event resource `calls`

Name	Value	Description
Mandatory response elements		
call	ID	The id of the conference whose elements have been updated.
Subscribable elements		
name	String	The name of the conference.
participants	Numeric	The number of participants currently in the conference.
distributedInstances	Numeric	The number of distributed instances of this conference that exist across the Call Bridge cluster (0 for an unclustered conference).
recording	active| inactive	One of: *active* - this conference is currently being recorded *inactive* - this conference is currently not being recorded.
endpointRecording	active| inactive	One of: *active* - this conference is currently being externally recorded by an endpoint (Lync client) *inactive* - this conference is currently not being externally recorded by an endpoint (Lync client).
streaming	active| inactive	One of: *active* - this conference is currently being streamed *inactive* - this conference is currently not being streamed.
lockState	locked| notLocked	One of: *locked* - this conference is currently being locked *notLocked* - this conference is currently not being locked.
callType	coSpace| forwarding| adHoc| lyncConferencing	One of: *coSpace* - this conference is the instantiation of a coSpace *forwarding* - this is a forwarded/" gateway" call *adHoc* - this is an ad hoc multi-party call *lyncConferencing* - this call leg is participating in a Lync conference.
callCorrelator	ID	The correlator GUID which is the same across all distributed instances of the call.

3 Call Bridge Groups and clusters

3.0.1 Call Bridge Groups and clusters

When a conference is hosted across a Call Bridge Group (or cluster), messages are passed between the Call Bridges in the group to ensure that each Meeting Server knows the active conferences and the roster list information for any conference for which it has one or more participants.

Subscriptions receive updates for local and remote participants to a conference, but when a remote participant leaves the conference, the parameter 'reason' won't be provided. Only subscriptions to the Call Bridge hosting the conference will receive the reason why the participant has left the conference. To receive information on all active conferences across the Call Bridge Group or cluster, the events client will need to subscribe to event resource **`calls`** on every Meeting Server, but to see event resources **`callInfo`** or **`callRoster`**, the events client only needs to subscribe to one of the Meeting Servers which reports on the conference.

When a Meeting Server is no longer participating in the conference due to participants hosted on its Call Bridge leaving the conference, the Call Bridge will deactivate the subscription, and the events client will have to pick another Meeting Server.

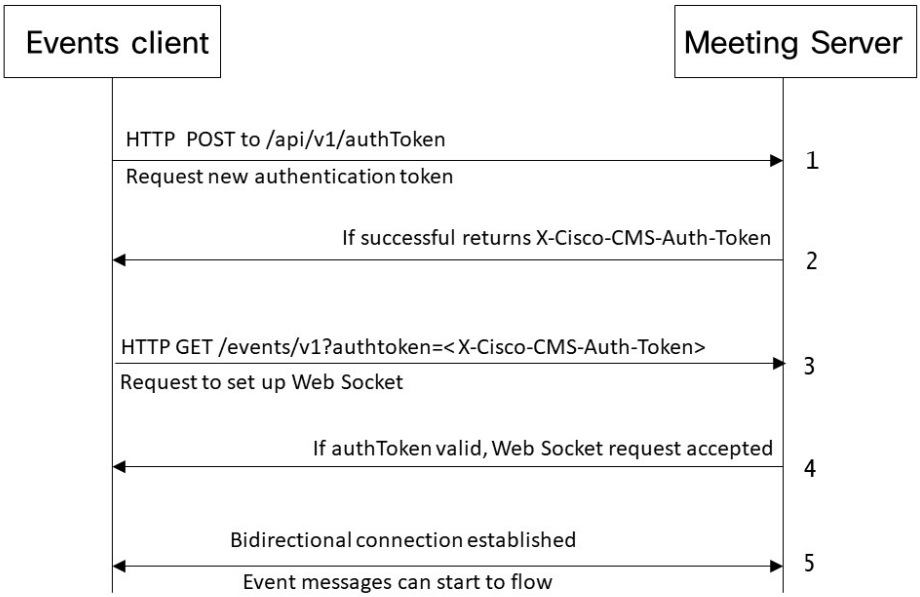
4 Example authentication flow

4.0.1 Example authentication flow

Before being allowed to subscribe to events on the Meeting Server, the events client needs to be authenticated by the Meeting Server and a connection using the WebSocket Protocol established between the events client and the Meeting Server. This connection enables the events client to receive event information without needing to constantly poll the API of the Meeting Server.

Figure 1 illustrates the call flow between the events client and the Meeting Server required to establish a WebSocket. You will need to use Python or a similar coding language for the events client to use to send the HTTP POSTs and GETs to the Meeting Server.

Figure 1: Overview of call flows establishing a WebSocket between the events client and the Meeting Server



1. The events client POSTs to /api/v1/authTokens in order to provision a new authorization token; for instance the request might look something like:

```
POST /api/v1/authTokens HTTP/1.1\r\n
Origin: http://xx.xxx.xxx.xxx:8080\r\n
Content-length: 0\r\n
Host: xx.xxx.xxx.xxx:8080\r\n
Accept: */*\r\n
Connection: keep-alive\r\n
Authorization: Basic Ym9iOmJ1aWxkZXI=\r\n
\r\n
```

2. The successful response from the Meeting Server, assuming that " authorization" relates to a user account with sufficient privilege, will be of the form:

   ```
   HTTP/1.1 200 OK\r\n
   X-Cisco-CMS-Auth-Token: 7174c102-61b3-47a6-8ff2-b86256cca958\r\n
   Connection: close\r\n
   \r\n
   ```

   The returned " X-Cisco-CMS-Auth-Token" is used in the next stage, the WebSocket connection itself

3. The client makes another HTTP request to set up the WebSocket connection, of the form:

   ```
   GET /events/v1?authToken=7174c102-61b3-47a6-8ff2-b86256cca958 HTTP/1.1\r\n/
   Host: xx.xxx.xxx.xxx\r\n/
   Connection: Upgrade\r\n/
   Pragma: no-cache\r\n/
   Cache-Control: no-cache\r\n/
   Upgrade: websocket\r\n/
   Origin: http://xx.xxx.xxx.xxx:8080\r\n/
   Sec-WebSocket-Version: 13\r\n/
   Accept-Encoding: gzip, deflate\r\n
   Accept-Language: en-GB,en-US;q=0.8,en;q=0.6\r\n
   Sec-WebSocket-Key: lGaahHe/KdA9lPdPxAIZfw==\r\n
   Sec-WebSocket-Extensions: permessage-deflate; client_max_window_bits\r\n
   \r\n
   ```

   Note: The X-Cisco-CMS-Auth-Token value should be sent to the Meeting Server as an " authToken" URI parameter. The Sec-WebSocket-Key value (and follow up Sec-WebSocket-Accept) values are as per RFC6455.

4. Assuming that the Meeting Server accepts the WebSocket connection (for instance, the authToken is valid) the success response will look like:

```
HTTP/1.1 101 Switching Protocols\r\n
Upgrade: websocket\r\n
Connection: upgrade\r\n
Sec-WebSocket-Accept: ZISmDfOsp675RM7TQKa0LbQKCqk=\r\n
\r\n
```

5. The bi-directional WebSocket connection is " ready to go" and the event messages can start to flow.

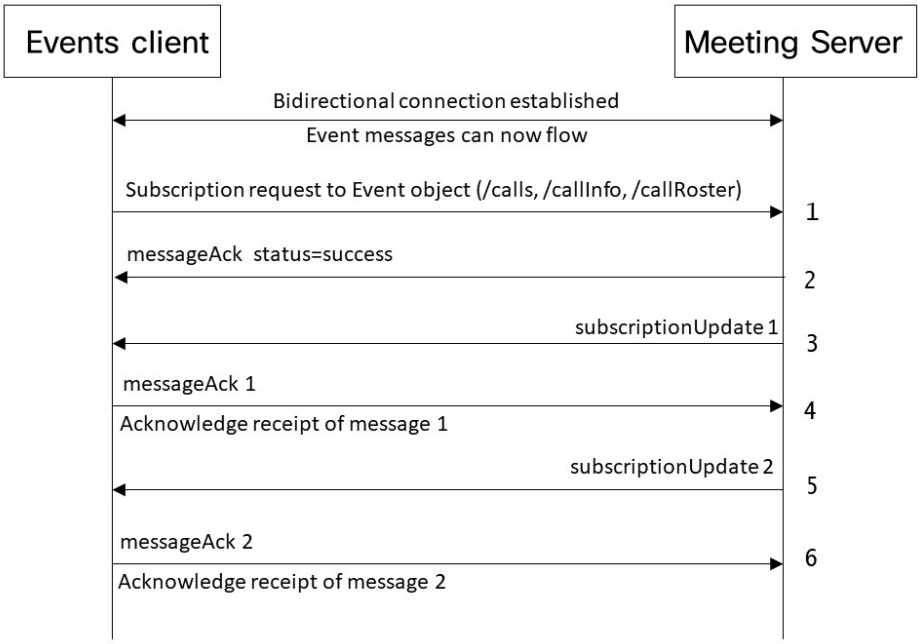
5 Example message flows

5.0.1 Example message flows

Once the WebSocket between the events client and the Meeting Server has been established, the events client can subscribe to the Meeting Server for updates on event types.

Figure 2 illustrates the call flow between the events client and the Meeting Server over the web socket connection. The subscriptions from the events client should be in the form of JSON files. For more detail on the call flow, see the explanations and examples following Figure 2.

Figure 2: Overview of event messages flowing between the events client and the Meeting Server



1. Initial subscription from the events client to the Meeting Server, requesting to subscribe to an event type, for example the calls list (active conference list).

```
{
     "type": "message",
     "message": {
          "messageId": 8,
          "type": "subscribeRequest",
          "subscriptions": [
               {
                    "index": 3,
                    "type": "calls",
                    "elements": [
                    "name",
                    "participants"
                    ]
               }
          ]
     }
}
```

In the above example the client is sending the subscribe request as its message with index 8, and is supplying a single resource to subscribe to the set (array) called subscriptions. It has used the index (tag) " 3" for this subscription, and its type is " calls" , which refers to the active calls list. Within this subscription, it has specified the set of elements in which it is interested, " name" (the conference name) and " participants" (the number of active participants). This set of elements has 2 main implications:

- it determines which changes trigger a Meeting Server → client update (in this case if the name or the participant count changes an update will be sent),
- it determines which elements are included in the Meeting Server → client update (in this case the record that will be sent to the client will include just the name and the participant count).

2. The Meeting Server responds with an ACK to the subscribe request:

```
{
     "type": "messageAck",
     "messageAck": {
          "messageId": 8,
          "status": "success"
     }
}
```

This ACK has a " messageId" of 8 to indicate to the client which message is being acknowledged, and a " status" code to show if the message has been successfully understood and acted upon.

3. The Meeting Server also keeps the requesting client updated as to the status of its active subscriptions. This is in the form of a subscriptionUpdate sent to the client, for instance:

```
{
     "type": "message",
     "message":
          "messageId": 1,
          "type": "subscriptionUpdate",
          "subscriptions": [
               {
                    "index": 3,
                    "state": "pending"
               }
          ]
     }
}
```

   This tells the client that its subscription with index " 3" is pending – it is in the process of being set up but is not yet active. The Meeting Server has tagged this message with a messageId of 1, and the client should ACK this message in order for the Meeting Server to send later updates.

4. The corresponding messageAck from the client to the Meeting Server looks like:

```
{
     "type": "messageAck",
     "messageAck": {
          "messageId": 1,
          "status": "success"
     }
}
```

   The Meeting Server and the client use their own individual number spaces for messageId values, which are not required to be distinct; neither are they required to be sequential.

5. Once the Meeting Server has successfully set up the requested subscription, it may send a further subscriptionUpdate, for example:

```
{
     "type": "message",
     "message": {
          "messageId": 2,
          "type": "subscriptionUpdate",
          "subscriptions": [
               {
                    "index": 3,
                    "state": "active"
               }
          ]
     }
}
```

This tells the client that the subscription (with index 3) is now active - the client can now expect to receive updates specific to that subscription. As before, this message from the Meeting Server needs to be acknowledged by the client.

6. Client acknowledges the subscription update for messageId 2. The client supplies messageId 2 in the ack, as that is the value used by the Meeting Server in the subscriptionUpdate.

```
{
     "type": "messageAck",
          "messageAck": {
          "messageId": 2,
          "status": "success"
     }
}
```

7. At some point after the client has subscribed to the calls list, the Meeting Server sends information on a conference just starting:

```
{
     "type": "message",
     "message": {
          "messageId": 3,
          "type": "callListUpdate",
          "subscriptionIndex": 3,
          "updates": [
               {
                    "call": "97c771ae-fc2e-4257-b129-30ee818e034b",
                    "updateType": "add",
                    "name": "Andy's coSpace",
                    "participants": 0
               }
          ]
```

```
    }
}
```

As before, this message, messageId 3, needs to be acked by the client.

Note: To avoid duplication, messageAcks are no longer shown in this example.

This update is tagged with subscriptionIndex " 3", indicating to the client which of its subscriptions the update refers to. The update includes a " type" to aid parsing of the data. The " updates" array contains the new information that the Meeting Server is supplying, in this example the " updateType" is an " add" which means that this is the first notification of this conference, and the conference in question has its GUID supplied in the " call" field. The " call" and " updateType" values appear in all updates, but the remaining fields are determined by the " elements" value supplied in the subscription request from the client. If the no " elements" node is supplied in the subscribe message (or it is empty) then no additional fields will be included. However, as " name" and " participants" were included in the example then those are present in the update.

8. After the initial " add" message for this conference, a participant joins the conference, and so a further update is received from the Meeting Server:

```
{
    "type": "message",
    "message": {
        "messageId": 4,
        "type": "callListUpdate",
        "subscriptionIndex": 3,
        "updates": [
            {
                "call": "97c771ae-fc2e-4257-b129-30ee818e034b",
                "updateType": "update",
                "participants": 1
            }
        ]
    }
}
```

The participants count is now " 1" , and the updateType is now " update" to show that this isn't the first message for the call in question (97c771ae-fc2e-4257-b129-30ee818e034b), but an update to the previous notification. As the " name" value for this call hasn't changed it isn't included in this update.

9. Once a client application has learnt about the presence of an active call in which it is interested (either via the events mechanism or through an API query or " callStart" CDR) the client application may then subscribe to resources specific to that call. For instance, it may re-configure its subscription with the Meeting Server with a new message such as:

```
{
     "type": "message",
     "message": {
          "messageId": 9,
          "type": "subscribeRequest",
          "subscriptions": [
               {
                    "index": 1,
                    "type": "callRoster",
                    "call": "97c771ae-fc2e-4257-b129-30ee818e034b",
                    "elements": [
                         "name",
                         "uri",
                         "state",
                         "importance"
                    ]
               },
               {
                    "index": 2,
                    "type": "callInfo",
                    "call": "97c771ae-fc2e-4257-b129-30ee818e034b",
                    "elements": [
                         "name",
                         "participants",
                         "streaming"
                    ]
               },
               {
                    "index": 3,
                    "type": "calls",
                    "elements": [
                         "name",
                         "participants"
                    ]
               }
          ]
     }
}
```

The client remains subscribed to " calls" (the active conference list) by keeping a subscription with the same " index" 3, in its set of " subscriptions" . However, it has now added subscriptions to " callRoster" and " callInfo" to the set. For these subscriptions, a specific " call" GUID needs to be supplied, in this case " 97c771ae-fc2e-4257-b129-

30ee818e034b" , which the client was notified of in an earlier " callListUpdate" message from the Meeting Server.

6 WebSocket specifications

This section details information on using WebSockets to establish a connection between an events client and the Meeting Server. Do not exceed the limits specified below:

- Maximum number of concurrent WebSocket connections per Meeting Server: 5
- Max number of simultaneous subscriptions per WebSocket connection: 100
- No support for fragmented WebSocket frames,
- No support for ping or pong control frames, or binary data frames.

Cisco Legal Information

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.

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.

All printed copies and duplicate soft copies of this document are considered uncontrolled. See the current online version for the latest version.

Cisco has more than 200 offices worldwide. Addresses and phone numbers are listed on the Cisco website at www.cisco.com/go/offices.

Cisco Trademark