Cisco Meeting Server

Cisco Meeting Server Release 3.3

Call Detail Records Guide

September 16, 2021

Contents

Figures:

Change History

Date	Change Summary
September 16, 2021	Updated for version 3.3.
May 12, 2021	Removed distributionLink from subType in callLegStart Record Contents.
April 09, 2021	Updated for version 3.2.
July 29, 2020	Updated for version 3.0, removed references to X-Series servers.
May 05, 2020	Clarification added to Section 4.6
April 08, 2020	Updated for version 2.9.
January 07, 2020	Minor correction
September 16, 2019	callMove and displayName missing from callLegUpdate record.
August 13, 2019	Changed title to "… 2.6 and later", no changes for version 2.7.
July 19, 2019	Minor correction
April 23, 2019	Updated for version 2.6. Added canMove, movedCallLeg and movedCallLegCallBridge to the callLegStart record.
December 12, 2018	Changed title to "… 2.4 and later", no changes for version 2.5.
September 20, 2018	Updated for version 2.4. Added endpointRecorded to the callEnd record.
December 20, 2017	Reissued for version 2.3. No additions or changes.
June 28, 2017	Added multiStreamVideo to mediaUsagePercentages in the callLegEnd records.
June 28, 2017	Added example on creating a CDR receiver.
May 03, 2017	Updated for version 2.2. Added ownerName field to callStart records..
December 20, 2016	Updated for version 2.1. Additions and changes are indicated.
August 03, 2016	Rebranded for Cisco Meeting Server 2.0

1 Introduction

The Cisco Meeting Server software can be hosted on specific servers based on Cisco Unified Computing Server (UCS) technology or on a specification-based VM server. Cisco Meeting Server is referred to as the Meeting Server throughout this document.

Note: Cisco Meeting Server software version 3.0 onwards does not support X-Series servers.

The Meeting Server generates Call Detail Records (CDRs) internally for key call-related events, such as a new SIP connection arriving at the server, or a call being activated or deactivated.

The server can be configured to send these records to a remote system to be collected and analyzed. There is no provision for records to be stored on a long-term basis on the Meeting Server, nor any way to browse CDRs on the Meeting Server itself.

The CDR system can be used in conjunction with the Meeting Server API, with the call ID and call leg IDs values being consistent between the two systems to allow cross referencing of events and diagnostics.

The Meeting Server supports up to four CDR receivers enabling you to deploy different management tools or multiple instances of the same management tool.

Note: Also refer to the Cisco Meeting Server API Reference guide.

1.1 How to Use this Document

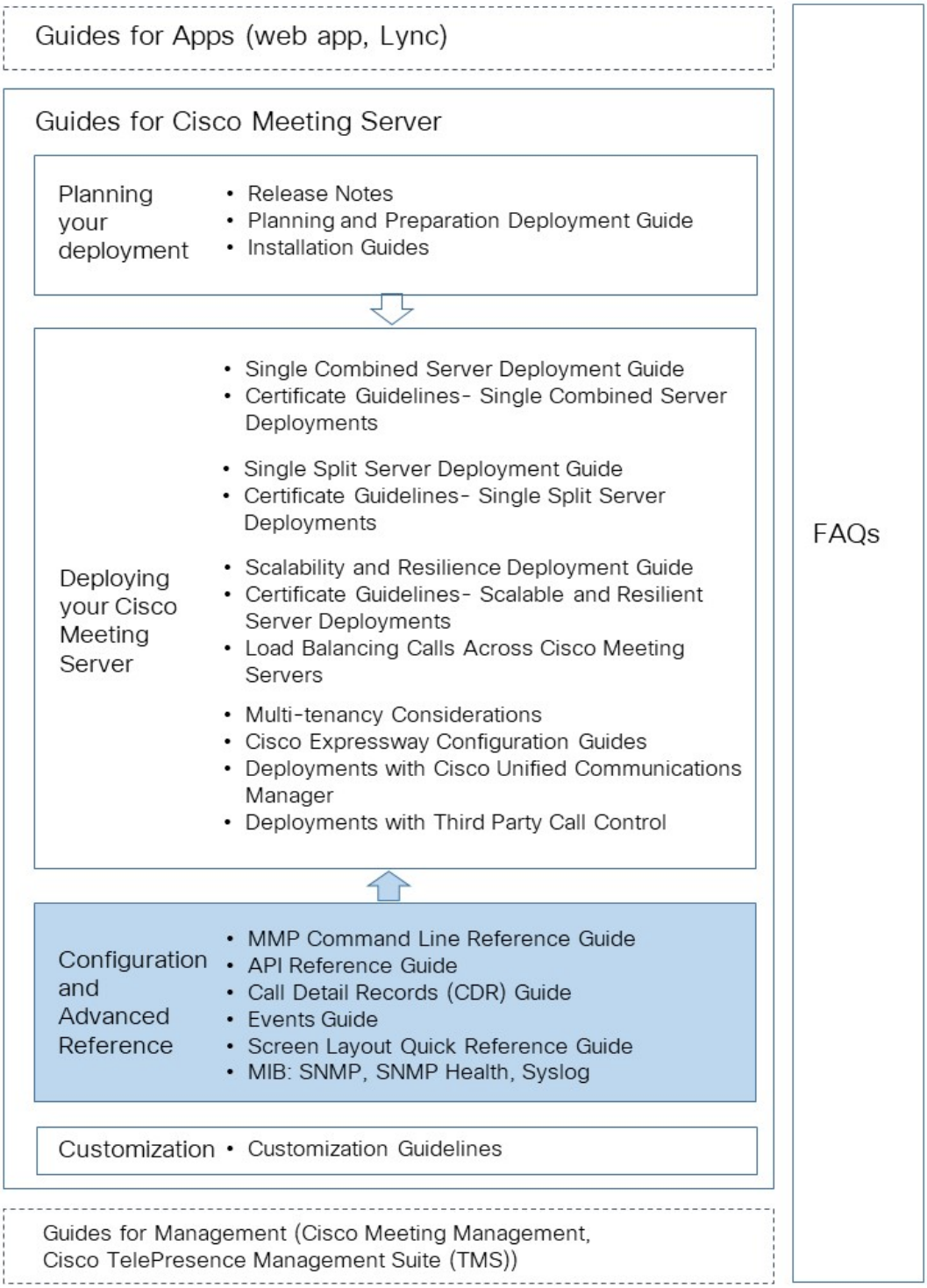
This document is one of a number of reference guides as shown in the figure below.

It is split into sections allowing you to build your knowledge by reading from front to back. In addition chapters 3, 4, and 5 act as a reference that can be “dipped into”. Each CDR record type and its fields are described in detail.

This document describes a “minimum set” of information; the XML nature of the records means that new elements may appear in new Call Bridge software versions and so you should always allow for this when parsing the records we generate. A receiver must be able to cope with additional, new elements, not mentioned in any existing version of the document (while at the same time we commit to providing what the document says we provide, according to the structure it describes).

These documents can be found on cisco.com.

Figure 1: Cisco Meeting Server documentation for release 3.3



2 General Mechanism

CDRs are sent out by the Meeting Server over HTTP or HTTPS as a series of XML documents. When new records are generated, a connection is made to the receiving system and the receiving system should expect to receive one or multiple records on this connection. When the Meeting Server has successfully sent a group of records to the receiver, those records are no longer stored by the Meeting Server, and responsibility for their long-term storage then moves to the receiving device. The Meeting Server considers the records to have been successfully sent to the receiver if the HTTP or HTTPS connection has been successfully established, the XML record data has been sent by the Meeting Server, and the receiver has acknowledged the data with a "200 OK" HTTP response.

The Call Bridge supports keepalive connections to allow it to send multiple (batches of) records on one TCP or TLS connection to a receiver.

Note: In scalable and resilient deployments where multiple Call Bridges act as a single entity, each Call Bridge provides CDRs for the call legs that it is running. Each CDR identifies the coSpace ID for the call leg. Then if a call is hosted over more than one Call Bridge, you can identify the same call on different Call Bridges by the same coSpace ID.

2.1 Configuring the Recipient Devices

Note: The list of CDR receivers is held locally to an individual call bridge, it is not stored in the database shared between clustered call bridges.

You can use either the Web Admin Interface or the API to configure the CDR receivers.

2.1.1 Using the Web Admin Interface to configure the CDR receivers

To configure the recipient of the CDRs:

1. Open the Web Admin Interface.
2. Go to **Configure > CDR** settings.
3. In the CDR Receiver Settings section, for each receiver, enter the receiver's HTTP or HTTPS URI (see Section 2.1.3).

2.1.2 Using the API to configure the CDR receivers

Use the following API objects to enable up to four CDR receivers to be configured for the Meeting Server:

- /system/cdrReceivers/
- /system/cdrReceivers/<CDR receiver id>

Issue a POST on the /system/cdrReceivers node to set the full URI of a new receiver. A GET request on /system/cdrReceivers shows the currently configured receivers.

Once a CDR receiver has been configured, its details can be read and updated by use of a GET or PUT on the /system/cdrReceivers/<CDR receiver id> node respectively. A CDR receiver can be removed by a DELETE of this node.

2.1.3 Recipient URI

The recipient URIs, as configured on the Meeting Server, can take one of a number of forms:

- http://monitoring.example.com/cdr_receiver1
  for a simple HTTP connection to TCP port 80 on the remote host “monitoring. example com”, to the URI “/cdr_receiver1”
- https://monitoring. example.com/cdr_receiver1
  As above, but using HTTPS, TCP port 443
- http://monitoring. example.com:8080/cdr_receiver1
  As above, but using TCP port 8080 instead of the default port number (80)
- http://monitoring. example.com/cdr_receiver1?system_id=cms1
  As above but supplying the parameter “system_id” with value “cms1” to the destination device. The Meeting Server will just send parameters as supplied in the URI field verbatim to the far end, and it is up to the receiving device to understand their meaning.

3 Record Types

CDRs are sent in XML as one or more "<record>" items within a parent "<records>" element. Each record has an associated "type" value that identifies what it describes, and determines which fields and attributes should be expected within it.

The encompassing “<records>” element includes:

- a “session” value that takes the form of a GUID that is unique for that session. The session GUID is created when the Call Bridge restarts; it will be the same for all active CDR receivers for a given running Call Bridge instance, but changes when that Call Bridge restarts. It is used by a receiving device to determine that the records it is receiving are being sent from the same session on the same device.
- a Call Bridge GUID if the Call Bridge is in a cluster. This identifies which Call Bridge in a cluster is sending the record. This remains the same across all restarts of the system. Note that it is not present in unclustered deployments. The Call Bridge GUID is the same as in the /callBridges API object.

The “<record>” items includes:

- a time value at which the record was generated on the Meeting Server. This timestamp is in RFC 3339 / ISO 8601 format, for instance “2014-02-28T16:03:25Z” for 4:03pm on 28 February 2014). Currently, the Meeting Server always supplies these times in UTC.
- the “correlatorIndex” that increments by 1 for each new record. Note: the combination of "session GUID" and "correlatorIndex" uniquely identifies a record across all receivers, enabling the receiver to determine whether it has received duplicate records.

  The “correlatorIndex” starts at “0” for the first record that the Call Bridge generates after booting up. The “correlatorIndex” for a record is the same across all CDR receivers. Hence for a receiver that is configured sometime after the Call Bridge has booted, the first record it receives may not be index 0.

  When a receiver successfully receives a record, it needs to send a “200 OK” HTTP response to the Call Bridge, the Call Bridge then sends the next set of records to the receiver. If the “200 OK” HTTP response is not successfully received by the Call Bridge, then the Call Bridge will resend the records resulting in the receiver receiving duplicate records.

  If a remote receiver has been unavailable for a period of time such that the Meeting Server has not been able to store all of the generated records internally, records pushed to the remote receiver will show a gap in the “correlatorIndex”.

- the "recordIndex” has been replaced by the “correlatorIndex”. The "recordIndex” is now deprecated and may be removed in future releases.

  For completeness, the following describes how to use the "recordIndex”.

The "recordIndex" can be used to detemine whether the Meeting Server has received duplicate records.

Note: If you have multiple cdr receivers then the "recordIndex" value can differ for the same record for different receivers.

This description assumes you only have one receiver. The "recordIndex" within a "<record>" items determines the sequence of records, starting at "1" for the first record that the Call Bridge generates, and increasing by 1 for each new record sent. This "recordIndex" value allows a CDR receiver to determine whether it has received duplicate records; the combination of session GUID value and recordIndex is unique. The Call Bridgee re-sends any CDRs for which it had not received a positive acknowledgement from the receiver (a "200 OK" HTTP response). If a receiver sends such a positive response but that response is not successfully received by the Call Bridge, the receiver may receive duplicate records – the "recordIndex" allows the receiver to detect this and not process the duplicate records.

If a remote receiver has been unavailable for a period of time such that the Meeting Server has not been able to store all of the generated records internally, records pushed to the remote receiver will include a numeric "numPreceedingRecordsMissing" value in the "<record>" tag. This signals to the remote receiver that this number of records (immediately preceding the record in whose header it appears) have been discarded and are no longer available. A CDR receiver should not see a discontinuity in the "recordIndex" sequence even in the presence of a non-zero value for "numPreceedingRecordsMissing".

The record types are described briefly in Table 1 below, and in more detail in Chapter 4.

Table 1: Overview of Record Types

Record type	Description
callStart	This record is generated when a call is either created or first instantiated from a coSpace. The record contains the call's ID, its name, and the ID of any associated coSpace.
callEnd	This record is generated when a call ends, and typically will be seen after the last call leg for the call has disconnected. The record contains the call ID, which should match a call ID in an earlier callStart record, and summary values for the call, such as the maximum number of call legs that were ever simultaneously active within the call.
callLegStart	This record is generated when a call leg is first created, because of an incoming connection, an outgoing call leg being established, or the user of a Cisco Meeting App connecting to a coSpace. The record contains the call leg ID, the remote party type (a SIP connection or a Cisco Meeting App device), the remote party "name" (for instance their SIP URI) and, if meaningful, whether the call leg was incoming or outgoing.

Record type	Description
callLegEnd	This record is generated when a call leg terminates, either because someone has chosen to disconnect or because another user with sufficient privileges has chosen to disconnect it. The record contains the call leg ID (which should correspond to a call leg ID from an earlier callLegStart record), the reason for the disconnection, and certain other summary fields relating to the lifetime of the call leg in question (which audio and video codecs were in use, for example).
callLegUpdate	This record is generated when a significant change occurs for a call leg, for instance that call leg being placed into a call, or (for the outgoing case) being answered and so transitioning to “connected” state.
recordingStart	This record is generated when recording starts on a call. The record contains the ID of the recording that is starting, the path where the recording will be stored (directory and filename), the URL of the recording device, the ID of the call that is being recorded and the ID of the call leg that is recording the call.
recordingEnd	This record is generated when the recording on a call is ended. It contains the ID of the recording that is ending.
streamingStart	This record is generated when streaming starts on a call. The record contains the ID of the streaming that is starting, the URL and stream name of the streaming, and the URL of the streaming device.
streamingEnd	This record is generated when the streaming on a call is ended. It contains the ID of the streaming that is ending.

4 Record Details

This section provides details of the parameter names and values which appear within the "<record>" tag for each record type.

4.1 callStart Record Contents

Parameter	Type	Description
id	ID	The ID of the call that is starting. This is conveyed as an "id" attribute within the "<call>" tag that encapsulates the callStart record.
name	String	The name of the call; typically this is the name of the coSpace if the call is associated with a coSpace.
coSpace	ID	The ID of the coSpace associated with this call. If this call is not associated with a coSpace (for instance, if it is an ad hoc call) then this field will not be present.
ownerName	String	Name of the owner of this call, taken from one of the following in descending priority: the value of the 'meetingScheduler' field of the coSpace, or the name of the user which owns the coSpace, or the jid of the user which owns the coSpace, or blank (this means that none of the above exist).
tenant	ID	In a multi-tenant deployment, specifies the tenant
cdrTag	String	If a coSpace was given a tag (see the API Reference), this is shown in the callStart CDR. The tag is an optional, up to 100 character text string used to help identify the call.
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 / "gateway" call

Parameter	Type	Description
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.
coSpaceMetaDataConfigured	true | false	This is set to true when metadata has been configured on the coSpace that this call is in. If the call is an adhoc call then this field is false. (From version 3.2)

Note: In distributed calls, if you see multiple overlapping "callStart" records for:

- a single coSpace ID, these call legs comprise a distributed coSpace call i.e. a coSpace call hosted by more than one Call Bridge. You can search for the coSpace ID using the API.
- a single callCorrelator value, these call legs comprise a distributed call. This can be a coSpace call but may not be; for example a “point-to-point call” in which each call leg is hosted by a different Call Bridge.

4.2 callEnd Record Contents

Parameter	Type	Description
id	ID	The ID of the call that is ending; an earlier “callStart” record will have been generated for the same call. This is conveyed as an “id” attribute within the “<call>” tag that encapsulates the callEnd record.
callLegsCompleted	Number	The number of call legs that have completed within this call.
callLegsMaxActive	Number	The maximum number of call legs that were simultaneously active within this call.
durationSeconds	Number	The length of time (in seconds) that this call was active for.
endpointRecorded	true|false	Has a value of true if the call has been recorded by an endpoint such as a Skype or Lync client at any given time. (From version 2.4)

4.3 callLegStart Record Contents

Parameter	Type	Description
id	ID	The ID of the call leg that is starting. This is conveyed as an “id” attribute within the “<callLeg>” tag that encapsulates the callLegStart record.
displayName	String	The "friendly name" for a SIP endpoint, a user's "real name" for an Cisco Meeting App connection, or what a user types for a web client guest connection. This value is blank if the far end does not provide a friendly name.
localAddress	String	Any local destination relevant to the call leg (e.g. what the caller connected to in order to reach the Meeting Server.) The interpretation of this value depends on the direction (see below). Therefore, this is the destination address for incoming calls, or the caller ID of outgoing calls. Note: In some call scenarios, no localAddress applies e.g. calling out to an Cisco Meeting App user from a coSpace with no defined URIs.
remoteAddress	String	For SIP calls, the remote URI relevant to the call leg. The interpretation of this value depends on the direction (see below). This is the destination URI for outgoing calls, or the source URI of incoming calls.
remoteParty	String	The address of the remote party of this call leg. For outgoing calls this is the output of the dial transform and may not contain a domain.
cdrTag	String	If a call leg was given a tag, this is shown in the CDR. The tag is an optional, up to 100 character text string used to help identify the call leg.
guestConnection	true|false	(optional) Connections known to be guest logins initiated via the WebRTC App have a value of True here.
recording	true|false	Connections known to be recording the call have a value of "true " here.
streaming	true|false	Connections known to be streaming the call have a value of "true " here
type	sip|acano	The type of call leg: either “acano” for a Cisco Meeting App connection or “sip” for a SIP connection.
subType	lync| avaya| lyncdistribution| webApp	A further specialization of the call leg type; if the call leg is “sip”, possible values here are "lync", "avaya", "lyncdistribution" or "webApp".

Parameter	Type	Description
lyncSubType	audioVideo| applicationSharing| instantMessaging	A further specialization of the call leg type if the call leg sub type is "lync". *audioVideo* – this is a Lync call leg used for exchange of audio and video between the Call Bridgeand Lync *applicationSharing* – this is a Lync call leg used for application or desktop sharing between Lync and the Call Bridge *instantMessaging* – this is a Lync call leg used for the exchange of instant messages between Lync and the Call Bridge
direction	incoming| outgoing	For both sip and "acano" call types: *incoming* –if the remote SIP device initiated the connection to the Meeting Server, *outgoing* – if the call leg was established from the Meeting Server to the remote SIP device.
call	ID	The call ID for this call leg. If known at the call leg start time, this may be included, otherwise it will be signaled in a later callLegUpdate record
ownerId	ID	The ID that a remote, managing, system has chosen to assign to this call leg, which has meaning only to that remote system. This field will be absent if no such owner ID has been assigned to this call leg.
sipCallId	String	If the call leg is a SIP connection, this field will hold the unique “Call-ID” value from the SIP protocol headers, if known at call leg start time.
groupId	ID	For Lync calls only, this parameter links the Presenter’s video callLeg and their presentation stream if they share content. This is also the ID that should be used when performing “participant” API operations that relate to this call leg. A Lync presentation can create an extra callLeg in the CDRs, and that these can be tied together using the groupId parameter. (The callId will of course be the same, but there can be other Lync call legs in the call that aren’t owned by the same user – it is the groupId that is unique to a Lync ‘connection’.) If the Lync caller stops and restarts sharing, there will be a different call leg ID for the content sharing connection that for the first presentation, but the groupID will be the same.

Parameter	Type	Description
replacesSipCallId	String	If the call leg replaces another SIP call, this field will hold the unique "Call-ID" value (as a string) from the SIP protocol headers of the call that was replaced.
canMove	true|false	Indicates whether the participant owning this call leg can be moved using the movedParticipant API command. (From version 2.6)
movedCallLeg	ID	If this call leg was created as part of a participant move, the ID is the GUID of that participant's call leg that it was moved from. (From version 2.6)
movedCallLegCallBridge	ID	If this call leg was created as part of a participant move, the ID is the GUID of the Call Bridge hosting the conference that the moved participant's call leg was homed on. (From version 2.6)
confirmationStatus	required | notRequired | confirmed	The value provided determines whether the participant owning the call leg has to confirm, or has already confirmed to join the call, as required by the call out being made with the confirmation=true parameter. (From version 3.2)

4.4 callLegEnd Record Contents

Parameter	Type	Description
id	ID	The ID of the call leg that is ending. This is conveyed as an “id” attribute within the “<callLeg>” tag that encapsulates the callLegEnd record.
cdrTag	String	If a call leg was given a tag, this is shown in the CDR. The tag is an optional, up to 100 character text string used to help identify the call leg.
reason		The reason that the call leg is ending (see the table in Section 5).
remoteTeardown	true|false	*true* - indicates that the ending of this call leg was initiated by the remote party *false* - indicates that the ending of this call leg was initiated by the Meeting Server

<table>
<tr><th>Parameter</th><th>Type</th><th>Description</th></tr>
<tr><td>encryptedMedia
unencryptedMedia</td><td></td><td>One or both of these values may be present to indicate the presence or absence (based on the value being “true” or “false”) of encrypted or unencrypted media during the lifetime of the call leg. If absent, that media type was not present for this call leg.</td></tr>
<tr><td>durationSeconds</td><td>Number</td><td>The length of time (in seconds) that this call leg was active for.</td></tr>
<tr><td>activatedDuration</td><td>Number</td><td>The length of time (in seconds) that this call leg was activated.</td></tr>
<tr><td>mediaUsagePercentages</td><td></td><td>Information on the percentage of this call leg's lifetime that the different types of media were active.The media types are:
mainVideoViewer – user was receiving main video
mainVideoContributor – user was contributing to main video
presentationViewer – user was receiving presentation information
presentationContributor – user was sharing a presentation
multistreamVideo – percentage of time that multistreamVideo was active.</td></tr>
<tr><td>multistreamVideo</td><td>Information on the transmitted multistream video during the lifetime of this call leg.
<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td>maxScreens</td><td>Number</td><td>The maximum number of multiscreen main video screens active during the lifetime of this call leg; for instance this will be 2 for dual video..</td></tr>
</table></td><td></td></tr>
</table>

<table>
<tr><th>Parameter</th><th>Type</th><th>Description</th></tr>
<tr><td>alarm</td><td colspan="2">There will be one or more of these elements if the call leg experienced any alarm conditions during its lifetime.
<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td>type</td><td>packetLoss|
excessiveJitter|
highRoundTripTime</td><td>packetLoss - packet loss was observed locally or reported by the far end for this call leg

excessiveJitter - high jitter values were observed locally or reported by the far end for this call leg

highRoundTripTime - a long round trip between the Meeting Server and the remote party was detected for this call leg</td></tr>
<tr><td>durationPercentage</td><td>Number</td><td>This value gives the percentage of the call duration for which thae alarm condition occurred.</td></tr>
</table></td></tr>
</table>

<table>
<tr><th>Parameter</th><th>Type</th><th>Description</th></tr>
<tr><td>rxAudio
txAudio</td><td colspan="2">Provides detail on the received audio (“rxAudio”,audio received by the Meeting Server from the remote party) and transmitted audio (“txAudio”) during the lifetime of this call leg. The rxAudio and txAudio sections may contain the following elements:
<table>
<tr><th>Parameter</th><th>Type</th><th>Description</th></tr>
<tr><td>codec</td><td>one of:
g711u
g711a
g722
g728
g729
g722_1
g722_1c
aac
speexNb
speexWb
speexUwb
isacWb
opus</td><td>the audio codec used:
g711u - G.711 mu law
g711a - G.711 a law
g722 - G.722
g728 - G.728
g729 - G.729
g722_1 - G.722.1
g722_1c - G.722.1C (G.722.1 Annex C)
aac - AAC
speexNb - Speex narrowband
speexWb - Speex wideband
speexUwb - Speex ultra-wideband
isacWb - iSAC (internet Speech Audio Codec) wideband
isacSwb - iSAC (internet Speech Audio Codec) superwideband</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>Parameter</th><th>Type</th><th>Description</th></tr>
<tr><td>rxVideo
txVideo</td><td colspan="2">Provides detail on the received video (“rxVideo”, video received by the Meeting Server from the remote party) and transmitted video (“txVideo”) during the lifetime of this call leg. The rxVideo and txVideo sections may contain the following elements:
<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td>codec</td><td>one of:
h261
h263
h263+
h264
h264Lync
vp8
rtVideo</td><td>the video codec used
h261 - H.261
h263 - H.263
h263+ - H.263+
h264 - H.264
h264Lync - H.264 SVC for Lync
vp8 - VP8
rtVideo - RTVideo</td></tr>
<tr><td>maxSizeWidth</td><td>Number</td><td>The width of the maximum video resolution used</td></tr>
<tr><td>maxSizeHeight</td><td>Number</td><td>The height of the maximum video resolution used</td></tr>
</table>
Note: If a “rxVideo” or “txVideo” section is absent, no video was sent in that direction.</td></tr>
<tr><td>ownerId</td><td>ID</td><td>The ID that a remote, managing, system has chosen to assign to this call leg, which has meaning only to that remote system.</td></tr>
<tr><td>sipCallId</td><td>String</td><td>If the call leg is a SIP connection, this field will hold the unique “Call-ID” value from the SIP protocol headers.</td></tr>
</table>

4.5 callLegUpdate Record Contents

Parameter	Type	Description
id	ID	The ID of the call leg that is being updated. This is conveyed as an “id” attribute within the “<callLeg>” tag that encapsulates the callLegUpdate record.
cdrTag	String	If a call leg was given a tag, this is shown in the CDR. The tag is an optional, up to 100 character text string used to help identify the call leg.
state	connected or value absent	If present, contains an indication of the call leg state; currently only the “connected” value is supported. An absence of this value indicates that the call leg has not yet reached the connected state.

Parameter	Type	Description
deactivated	true|false	Indicates if the call leg is currently deactivated
remoteAddress	String	For SIP calls, the remote uri relevant to the call leg. The interpretation of this value depends on the direction (see below). Therefore, this is the destination uri for outgoing calls, or the source uri of incoming calls.
call ivr		The call ID for this call leg, or the (empty) “ivr” indication if the call leg is currently in an IVR.
ownerId	ID	The ID that a remote, managing, system has chosen to assign to this call leg, which has meaning only to that remote system.
sipCallId	String	If the call leg is a SIP connection, this field will hold the unique “Call-ID” value from the SIP protocol headers, if known at call leg start time.
groupID	ID	For Lync calls only, this parameter links the Presenter’s video callLeg and the Presentation stream being sent.
displayName	String	The "friendly name" for a SIP endpoint, a user's "real name" for an Cisco Meeting App connection, or what a user types for a web client guest connection. This value is blank if the far end does not provide a friendly name.
canMove	true|false	Whether the participant owning this call leg can be moved using the movedParticipant API command.
confirmationStatus	required | notRequired | confirmed	The value provided determines whether the participant owning the call leg has to confirm, or has already confirmed to join the call, as required by the call out being made with the confirmation=true parameter. (From version 3.2)

The callLegUpdate record is sent by the Meeting Server when any of the call leg characteristics it refers to change for a call leg. For example, a CDR receiver would expect to see such an update record if a call leg moves from an IVR into a call, or if an external management system changed the “ownerId” associated with that call leg.

4.6 recordingStart Record Contents

Parameter	Type	Description
id	ID	The ID of the recording that is starting. This is conveyed as an “id” attribute within the “<recording>” tag that encapsulates the recordingStart record.
path	String	A string holding the directory and filename of the recording. (Applicable to internal XMPP recorder only.)

Parameter	Type	Description
recorderUri	String	The URI of the recording device if it is a SIP recorder. (Applicable to external third-party SIP recorder only.)
call	ID	The ID of the call that is being recorded.
callLeg	ID	The ID of the call leg that is recording the call.

4.7 recordingEnd Record Contents

Parameter	Type	Description
id	ID	The ID of the recording that is ending. This is conveyed as an “id” attribute within the “<recording>” tag that encapsulates the recordingEnd record.

4.8 streamingStart Record Contents

Parameter	Type	Description
id	ID	The ID of the streaming that is starting. This is conveyed as an “id” attribute within the “<streaming>” tag that encapsulates the streamingStart record.
streamerUri	URL	The URL of the streaming device. (Applicable to the internal SIP streamer component.)
call	ID	The ID of the call that is being streamed.
callLeg	ID	The ID of the call leg that is streaming the call.

4.9 streamingEnd Record Contents

Parameter	Type	Description
id	ID	The ID of the streaming that is ending. This is conveyed as an “id” attribute within the “<streaming>” tag that encapsulates the streamingEnd record.

5 Reason Codes in Call Leg End Records

Call leg end records contain a reason code (within a “<reason>” tag) and a separate indication of whether the Meeting Server or the remote party chose to disconnect that call leg (a “<remoteTeardown>” section containing either “true” or “false”).

Although the party which caused the disconnection can be determined by the disconnect reason, a separate remote or local teardown indication allows future-proofing to the extent that if new reason codes are added that are not understood by a CDR receiver, basic knowledge of which side initiated the disconnect can still be obtained.

The possible values for the “<reason>” code are:

Reason	Description
apiInitiatedTeardown	The call leg was disconnected by the Meeting Server in response to an API request to do so
callDeactivated	The call leg was disconnected by the Meeting Server because the call of which it was part was deactivated, and the deactivate action for the call leg was set to "disconnect". See the API Reference for details
callEnded	The call leg was disconnected by the Meeting Server because the call it was part of ended, for instance in response to an API command to destroy it
callMoved	The call leg was moved to improve the efficiency in the use of the Call Bridge resources
clientInitiatedTeardown	The call leg was disconnected by the Meeting Server in response to a request to do so by an Cisco Meeting App with sufficient privileges
confirmationTimeOut	The call leg was disconnected because the remote destination did not respond in time. The voice prompt "you've been invited to a call, press 1 to join" will have been played, but the person on the other end did not press a key within a minute, causing the call leg to be disconnected using this reason code.
dnsFailure	A failure to resolve the host name of a remote destination; for example, as part of the process of establishing a connection to a remote system
encryptionRequired	The call leg was disconnected because there was a requirement for encrypted media that was not able to be met
error	An error has occurred during a SIP call resulting in the disconnection of the call leg. This may be caused by the SIP endpoint losing power or crashing during a call. If this happens repeatedly then turn on SIP tracing.
incorrectPasscode	After the maximum number of retries, the user did not supply the correct PIN for the call or coSpace they wanted to join

Reason	Description
ivrTimeout	The call leg connected to an IVR but was not able to be transitioned to a call within the required time
ivrUnknownCall	After the maximum number of retries, the user did not supply a valid call ID to join when in the IVR
localTeardown	A normal teardown of the call leg by the Meeting Server
participantLimitReached	You tried to add a new participant beyond the maximum number allowed for the call
remoteBusy	The call leg disconnected because the remote party signaled that they were busy and unable to accept the connection
remoteRejected	The call leg was rejected by the remote party
remoteTeardown	The call leg was disconnected by the remote party
ringingTimeout	The call leg reached the remote device, which rang but was not answered within the required time interval
tenantParticipantLimitReached	You tried to add a new participant beyond the maximum number allowed for the owning tenant
timeout	The call leg was disconnected by the Meeting Server because of a protocol timeout, for instance a SIP session timeout or the lack of a mandatory response to a SIP request
unknownDestination	The call leg was an incoming connection to a destination that did not resolve to a valid coSpace or user

6 Example Traffic Flow

The following trace shows a typical example traffic flow. It covers two SIP clients connecting to a meeting, then one ending the meeting, and the other SIP call then being dropped. The XML in this example has be formatted to make it easier to read.

Events post #1

```
<?xml version="1.0"?>
<records session="a865433a-4926-4549-a701-9bb5b93c75e6"
callBridge="158ba4f7-70eb-4a35-982c-71d4f1674277">
  <record type="callLegStart" time="2015-07-23T07:32:55Z" recordIndex="1"
correlatorIndex="0">
    <callLeg id="fc9c85ca-8c41-4a1a-9252-b16977d1e4e1">
      <remoteParty>sipclient1@example.com</remoteParty>
      <localAddress>access1@127.0.0.1</localAddress>
      <type>sip</type>
      <direction>incoming</direction>
      <groupId>18da80f3-8a71-4255-aa90-e1677b99b588</groupId>
      <sipCallId>b8a81da5-c24c-43db-ba58-742f587faec8</sipCallId>
    </callLeg>
  </record>
</records>
```

Events post #2

```
<?xml version="1.0"?>
<records session="a865433a-4926-4549-a701-9bb5b93c75e6"
callBridge="158ba4f7-70eb-4a35-982c-71d4f1674277">
  <record type="callStart" time="2015-07-23T07:32:55Z" recordIndex="2"
correlatorIndex="1">
    <call id="46d49cb4-8171-4abc-97f5-b88035b1da0a">
      <name>test564_1</name>
      <callType>coSpace</callType>
      <coSpace>50605235-60cf-484a-9fa1-278ad0646243</coSpace>
      <callCorrelator>5f3300c5-ca67-40e0-a503-
91baec70dbbe</callCorrelator>
    </call>
  </record>
  <record type="callLegUpdate" time="2015-07-23T07:32:55Z"
recordIndex="3" correlatorIndex="2">
    <callLeg id="fc9c85ca-8c41-4a1a-9252-b16977d1e4e1">
      <state>connected</state>
      <call>46d49cb4-8171-4abc-97f5-b88035b1da0a</call>
      <groupId>18da80f3-8a71-4255-aa90-e1677b99b588</groupId>
      <sipCallId>b8a81da5-c24c-43db-ba58-742f587faec8</sipCallId>
    </callLeg>
  </record>
</records>
```

Events post #3

```
<?xml version="1.0"?>
<records session="a865433a-4926-4549-a701-9bb5b93c75e6"
callBridge="158ba4f7-70eb-4a35-982c-71d4f1674277">
  <record type="callLegStart" time="2015-07-23T07:32:55Z" recordIndex="4"
correlatorIndex="3">
    <callLeg id="9cfdb064-3ae9-4b08-a003-6478187f375f">
      <remoteParty>sipclient2@example.com</remoteParty>
      <localAddress>access2@127.0.0.1</localAddress>
      <type>sip</type>
      <direction>incoming</direction>
      <groupId>3420c93f-f33c-4c9e-be95-d0d1bfb207f0</groupId>
      <sipCallId>a939937c-8b5e-4376-92de-97635983d7ef</sipCallId>
    </callLeg>
  </record>
</records>
```

Events post #4

```
<?xml version="1.0"?>
<records session="a865433a-4926-4549-a701-9bb5b93c75e6"
callBridge="158ba4f7-70eb-4a35-982c-71d4f1674277">
  <record type="callLegUpdate" time="2015-07-23T07:32:55Z"
recordIndex="5" correlatorIndex="4">
    <callLeg id="9cfdb064-3ae9-4b08-a003-6478187f375f">
      <state>connected</state>
      <call>46d49cb4-8171-4abc-97f5-b88035b1da0a</call>
      <groupId>3420c93f-f33c-4c9e-be95-d0d1bfb207f0</groupId>
      <sipCallId>a939937c-8b5e-4376-92de-97635983d7ef</sipCallId>
    </callLeg>
  </record>
</records>
```

Events post #5

```
<?xml version="1.0"?>
<records session="a865433a-4926-4549-a701-9bb5b93c75e6"
callBridge="158ba4f7-70eb-4a35-982c-71d4f1674277">
  <record type="callLegEnd" time="2015-07-23T07:33:05Z" recordIndex="6"
correlatorIndex="5">
    <callLeg id="9cfdb064-3ae9-4b08-a003-6478187f375f">
      <reason>remoteTeardown</reason>
      <remoteTeardown>true</remoteTeardown>
      <durationSeconds>10</durationSeconds>
      <mediaUsagePercentages>
        <mainVideoViewer>100.0</mainVideoViewer>
        <mainVideoContributor>100.0</mainVideoContributor>
      </mediaUsagePercentages>
      <unencryptedMedia>true</unencryptedMedia>
      <rxAudio>
        <codec>g722</codec>
        <packetStatistics>
```

```
            <packetLossBursts>
              <duration>0.000</duration>
              <density>0.00</density>
            </packetLossBursts>
            <packetGap>
              <duration>9.701</duration>
              <density>0.00</density>
            </packetGap>
          </packetStatistics>
        </rxAudio>
        <txAudio>
          <codec>g722_1c</codec>
        </txAudio>
        <rxVideo>
          <codec>h264</codec>
          <maxSizeWidth>768</maxSizeWidth>
          <maxSizeHeight>448</maxSizeHeight>
          <packetStatistics>
            <packetLossBursts>
              <duration>0.000</duration>
              <density>0.00</density>
            </packetLossBursts>
            <packetGap>
              <duration>8.597</duration>
              <density>0.00</density>
            </packetGap>
          </packetStatistics>
        </rxVideo>
        <txVideo>
          <codec>h264</codec>
          <maxSizeWidth>1280</maxSizeWidth>
          <maxSizeHeight>720</maxSizeHeight>
        </txVideo>
        <sipCallId>a939937c-8b5e-4376-92de-97635983d7ef</sipCallId>
      </callLeg>
    <record>
</records>
```

Events post #6

```
<?xml version="1.0"?>
<records session="a865433a-4926-4549-a701-9bb5b93c75e6"
callBridge="158ba4f7-70eb-4a35-982c-71d4f1674277">
  <record type="callLegEnd" time="2015-07-23T07:33:05Z" recordIndex="7"
correlatorIndex="6">
    <callLeg id="fc9c85ca-8c41-4a1a-9252-b16977d1e4e1">
      <reason>callDeactivated</reason>
      <remoteTeardown>false</remoteTeardown>
      <durationSeconds>10</durationSeconds>
      <mediaUsagePercentages>
        <mainVideoViewer>100.0</mainVideoViewer>
        <mainVideoContributor>100.0</mainVideoContributor>
      </mediaUsagePercentages>
```

```
      <unencryptedMedia>true</unencryptedMedia>
      <rxAudio>
        <codec>g711u</codec>
        <packetStatistics>
          <packetLossBursts>
            <duration>0.000</duration>
            <density>0.00</density>
          </packetLossBursts>
          <packetGap>
            <duration>9.702</duration>
            <density>0.00</density>
          </packetGap>
        </packetStatistics>
      </rxAudio>
      <txAudio>
        <codec>g722_1c</codec>
      </txAudio>
      <rxVideo>
        <codec>h264</codec>
        <maxSizeWidth>1280</maxSizeWidth>
        <maxSizeHeight>720</maxSizeHeight>
        <packetStatistics>
          <packetLossBursts>
            <duration>0.000</duration>
            <density>0.00</density>
          </packetLossBursts>
          <packetGap>
            <duration>8.484</duration>
            <density>0.00</density>
          </packetGap>
        </packetStatistics>
      </rxVideo>
      <txVideo>
        <codec>h264</codec>
        <maxSizeWidth>1024</maxSizeWidth>
        <maxSizeHeight>576</maxSizeHeight>
      </txVideo>
      <sipCallId>b8a81da5-c24c-43db-ba58-742f587faec8</sipCallId>
    </callLeg>
  </record>
  <record type="callEnd" time="2015-07-23T07:33:05Z" recordIndex="8"
correlatorIndex="7">
    <call id="46d49cb4-8171-4abc-97f5-b88035b1da0a">
      <callLegsCompleted>2</callLegsCompleted>
      <callLegsMaxActive>2</callLegsMaxActive>
      <durationSeconds>10</durationSeconds>
    </call>
  </record>
</records>
```

Appendix A Example script for creating a CDR receiver

The following python script illustrates how to create a CDR receiver. The example is for illustrative purposes only, and Cisco will not provide any support or warranty in the use of the code. Cisco reserves copyright of the code.

```
#!/usr/bin/python

## Example CDR receiver code for Cisco Meeting Server
## Copyright - Cisco Systems (2013-2017)
## No support, warranty or liability exists for this code



import BaseHTTPServer
import sys
import getopt
import ssl

class RequestHandler(BaseHTTPServer.BaseHTTPRequestHandler):
     handler = BaseHTTPServer.BaseHTTPRequestHandler
     handler.protocol_version = 'HTTP/1.1'
     print "using protocol version:", handler.protocol_version

     def do_GET(self) :
          #print 'received request for GET', self.path
          self.send_response(200)
          self.end_headers()
     def do_POST(self) :
          print 'received request for POST', self.path
          length = int(self.headers['Content-Length'])
          post_data = self.rfile.read(length)
          print 'data:', post_data
          self.send_response(200)
          self.end_headers()
     def log_message(self, format, *args):
          return

def main(argv) :
     try:
```

```
            opts, args = getopt.getopt(argv, 'p:c:k:')
            port = [val for opt,val in opts if opt=='-p'][0]
            assert(len(port) > 0)
            certfile_name = ''
            keyfile_name = ''
            for opt,val in opts :
                  if opt=='-c' :
                        certfile_name = val
                  if opt=='-k' :
                        keyfile_name = val
      except:
            print 'usage: cdr_receiver.py -p <port> [-c <certfile path>] [-
            k <keyfile path>]'
            sys.exit(2)
      server_address = ('', int(port))
      httpd = BaseHTTPServer.HTTPServer(server_address, RequestHandler)
      if (len(certfile_name) > 0) :
            print 'HTTPS mode with certfile', certfile_name
            httpd.socket = ssl.wrap_socket (httpd.socket, keyfile=keyfile_
            name, certfile=certfile_name, server_side=True)
      try :
            httpd.serve_forever()
      except KeyboardInterrupt:
            pass
      httpd.server_close()

if __name__ == "__main__":
      main(sys.argv[1:])
```

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