**Go to: Call Flow Details for MediaSense**

| Introduction |
| --- |
| This page lists frequently asked questions about Cisco MediaSense. |
| If you want to **ask a question**, **post a comment**, or **discuss an issue** regarding a specific article, then follow this procedure:<br>1. Log in<br>2. Navigate to the article in question<br>3. Click the **Discussion** tab at the top of the page<br>4. Edit the Discussion page as described above under Editing<br>5. To start a different thread under a new subheading on the Discussion page, click the **+** tab at the top of the page instead of the Edit tab. |

# Contents

# How do you correlate reference call IDs for different call scenarios, under Unified Communications Manager phone forking?

In Cisco MediaSense, the meta data for each call only provides the xRefCi (reference call ID) and the device ref (extension) of the forking device and the far-end device (can be a conference bridge or any other phone).

The xRefCi parameter is Unified Communication Manager's identifier for a particular media stream. They do not always correspond 1:1 with the recorded tracks.

## MediaSense Search and Play Call Association Feature

MediaSense generates multiple sessions for a call being recorded in case of hold/resume or transfer, which makes it difficult for users to identify all the recording sessions in a call. To enable users to associate these recording sessions in a single call, MediaSense introduces a new feature termed as Call Association. Through this feature, all the strongly associated calls having a common xRefci value are grouped together. MediaSense 10.5 supports the call association feature for Built-in-Bridge recordings.

## Agent Hold/Resume Scenario

1. Agent A (extn 1000) and Caller C (extn 2000) are talking to each other
2. Agent A puts call on hold
3. Agent A resumes call

There are two recording sessions for this scenario:

- Session with sessionId = S1 with following two tracks, for the time period / segment before agent puts call on hold.
  trackNumber = 0 with participant A (deviceRef = 1000, xRefCi = aaaa)
  trackNumber = 1 with participant B (deviceRef = 2000, xRefCi = cccc)
- Session with sessionId = S2 with following two tracks, for the time period / segment after agent resumes the call.
  trackNumber = 0 with participant A( deviceRef = 1000, xRefCi = aaaa)
  trackNumber = 1 with participant B (deviceRef = 2000, xRefCi = cccc)

MediaSense does not record the the segment of the call while agent has put the call on hold.

## Customer Hold/Resume Scenario

1. Agent A (extn 1000) and Caller C (extn 2000) are talking to each other
2. Customer C puts call on hold
3. Customer C resumes call

The entire call is recorded in one session for this scenario:

- Session with sessionId = S1 with following two tracks
  trackNumber = 0 with participant A (deviceRef = 1000, xRefCi = aaaa)
  trackNumber = 1 with participant B (deviceRef = 2000, xRefCi = cccc)

In this scenario MediaSense also records the the segment of the call while customer has put the call on hold.

## Agent Transfer to Another Agent Scenario

### Overview

1. Caller C (extn 2000) calls Agent A (extn 1000)
2. Agent A (extn 1000) consults Agent B (3000)
3. Agent A (extn 1000) completes transfer.
4. Agent B (extn 3000) hangs up.

**In case of Unified Communications Manager 9.x and earlier, summarized results are**

1. Caller C (extn 2000) calls Agent A (extn 1000)

   Session S1 STARTED, Track 0 is A (extn 1000), Track 1 is C (extn 2000)

2. Agent A (extn 1000) transfers call to another agent B (extn 3000). Both A and B devices are set up for forking.

   C (extn 2000) hearing MoH
   A (extn 1000) talking to B (extn 3000)

   - Session S1 ENDED
   - Session S2 STARTED, Track 0 is A (extn 1000), Track 1 is B (extn 3000)
   - Session S3 STARTED, Track 0 is B (extn 3000), Track 1 is A (extn 1000)

   **Notes:**

   - Session S1 ends because Agent A phone has placed Caller C on hold.
   - Sessions S2 and S3 exist because both phones are configured for forking.
   - The participants as well as the xRefCi for the two participants in S2 and S3 are identical, but in reverse positions from each other.
   - xRefCi values of S1 are not reflected in either S2 or S3, since the consult is considered an independent call.
3. Agent A (extn 1000) completes transfer.

   C (extn 2000) talking to B (extn 3000)
   A (extn 1000) disconnected

   - Session S2 ENDED.
   - Session S3 UPDATED, Track 0 is B (extn 3000), Track 1 is C (extn 2000).

   **Notes:**

   - A far end transfer triggers an UPDATE of the existing recording session.
   - The far-end participant changes to that of S1.
   - S3 new far-end xRefCi matches S1 far-end xRefCi.
4. Agent B (extn 3000) hangs up.

   C (extn 2000) and B (extn 3000) are disconnected.
   Session S3 ENDED
   **Notes:**
   A far end transfer results in an update of the existing session. The forking phone remains the only participant in Track 0, but the participant in Track 1 changes to the new party.

Metadata for sessions in case of Agent to Agent Transfer in Communications Manager 9.x and earlier

**In case of Unified Communications Manager 10.0 and later, summarized results are**

1. Caller C (extn 2000) calls Agent A (extn 1000)

   C (extn 2000) talks to Agent A (extn 1000)
   Session S1 - STARTED - Track 0 is A (extn 1000), Track 1 is C (2000)

2. Agent A (extn 1000) consults Agent B (extn 3000)

   C (extn 2000) hearing MoH  A (extn 1000) talking to B (extn 3000)

   - Session S1 - ENDED
   - Session S2 - STARTED - Track 0 is A (extn 1000), Track 1 is B (extn 3000)
   - Session S3 - STARTED - Track 0 is B (extn 3000), Track 1 is A (extn 1000)

   **Notes:**

   - Session S1 ends because Agent A phone has placed Caller C on hold.
   - Sessions S2 and S3 both exist because both phones are configured for forking.
   - The participants as well as the xRefCi for the two participants in S2 and S3 are identical but in reversed positions from each other.
   - S1 xRefCi values are not reflected in either S2 or S3, since the consult is considered an independent call

3. Agent A (extn 1000) completes transfer.

   C (extn 2000) talking to B (extn 3000)
   A (extn 1000) disconnected

   - Session S3 - ENDED
   - Session S2 - ENDED
   - Session S4 - STARTED - Track 0 is B (extn 3000), Track 1 is C (extn 2000)

   **Notes:**

   - A far end transfer triggers the end of one recording session and the start of another recording session.
   - Though a new session starts, its xRefCi values will match the previous sessions.
   - S4 far-end xRefCi matches S1 far-end xRefCi, and S4 near-end xRefCi matches S3 near-end xRefCi.

4. Agent B (extn 3000) hangs up.

   C (extn 2000) and B (extn 3000) disconnected
   Session S4 - ENDED
   **Notes:** A far end transfer results in the end of one recording session and the start of another recording session.

Metadata for sessions in case of Agent to Agent Transfer in Communications Manager 10.x and later

## Agent Conference with Another Agent Scenario

### Overview

1. Caller C (extn 2000) calls Agent A (extn 1000)
2. Agent A (extn 1000) consults Agent B (extn 3000)
3. Agent A (extn 1000) completes conference.
4. Agent A (extn 1000) drops from conference.
5. Agent B (extn 3000) hangs up.

**In case of Unified Communications Manager 9.x and earlier, summarized results are**

1. Caller C (extn 2000) calls Agent A (extn 1000)

   C (extn 2000) talking to A (extn 1000)
   Session S1 STARTED - Track 0 is A (extn 1000), Track 1 is C (extn 2000)

2. Agent A (extn 1000) consults Agent B (extn 3000)

   C (extn 2000) hearing MoH  A (extn 1000) talking to B (extn 3000)

   - ◊ Session S1 ENDED
   - ◊ Session S2 STARTED - Track 0 is A (extn 1000), Track 1 is B (extn 3000)
   - ◊ Session S3 STARTED - Track 0 is B (extn 3000), Track 1 is A (extn 1052

   **Notes:**

   - ◊ Session S1 ends because Agent A phone has placed Caller C on hold.
   - ◊ Sessions S2 and S3 exist because both phones are configured for forking.
   - ◊ The participants as well as the xRefCi for the two participants in S2 and S3 are identical but in reversed positions from each other.
   - ◊ S1 xRefCi values are not reflected in either S2 or S3, since the consult is considered an independent call

3. Agent A (extn 1000) completes conference.

   C (extn 2000) talking to A (extn 1000) and B (extn 3000)

   - ◊ Session S2 ENDED
   - ◊ Session S3 UPDATED - Track 0 is B (ext 3000), Track 1 is Conference Bridge
   - ◊ Session S4 STARTED - Track 0 is A (extn 1000), Track 1 is Conference Bridge

   **Notes:**
   A far end transfer triggers an UPDATE of the existing recording session.
   The completion of a conference is implemented as follows:
   During the consult:

   - ◊ The consulting phone has a primary call on hold and an active consult call.
   - ◊ The consulted phone only has one active call (the consult call)

   When the conference is completed (all parties connected):

   - ◊ The consulting phone's consult call terminates.
   - ◊ The consulting phone's primary call gets a far end transfer to the conference bridge.
   - ◊ The consulted phone gets a far end transfer to the conference bridge.

As a result:

- ◊ S2 terminates, because it represents the consulting phone's consult call, which also terminates.
- ◊ S4 starts; it represents the continuation and far end transfer of A's primary call, but the original S1 cannot be UPDATED because was previously terminated due to earlier hold.
- ◊ S3 gets UPDATED because B's far end is simply being transferred from A to the conference bridge.
- ◊ S4 near-end xRefCi value will match S1 near-end xRefCi value.

4. Agent A (extn 1000) drops from conference.

A (extn 1000) disconnected. C (extn 2000) talking to B (extn 3000) Session S3 UPDATED - Track 0 is B (extn 3000), Track 1 is C (extn 2000)

**Notes:**

- ◊ The de-escalation of a conference into a normal two-party call is implemented as both remaining phones' far end transferring to each other
- ◊ A far end transfer triggers an UPDATE of the existing recording session.
- ◊ S3 and S1 will have matching near-end xRefCi values. Note that only one session remains active because Caller C does not have forking enabled.

5. Agent B (extn 3000) hangs up.

C (extn 2000) and B (extn 3000) disconnected.
Session S4 ENDED

**Notes:**

- ◊ A far end transfer results in an update of the existing session. The forking phone remains the only participant in Track 0, but the participant in Track 1 changes to the new party.
- ◊ A conference is created by transferring all phones to the conference bridge. Therefore, a conference acts just like a set of transfers. Existing sessions are updated; in those sessions the forking phone remains the only participant in Track 0, but the participant in Track 1 changes to the conference bridge.
- ◊ Once the third party drops from the conference, the parties are transferred to each other. This updates the existing sessions again; the forking phone remains the only participant in Track 0, but the participant in Track 1 changes to the other party.
- ◊ If a fourth party is added to the conference bridge, there is no indication in the metadata, unless the fourth party also has its own forking enabled.

Metadata for sessions in case of Unified Communications Manager 9.x and earlier

**In case of Unified Communications Manager 10.x and later, summarized results are**

1. Caller C (extn 2000) calls Agent A (extn 1000)

C (extn 2000) talking to A (extn 1000) Session S1 - STARTED - Track 0 is A (extn 1000), Track 1 is C (extn 2000)

2. Agent A (extn 1000) consults Agent B (extn 3000)

C (extn 2000) hearing MoH A (extn 1000) talking to B (extn 3000)

- ◊ Session S1 - ENDED

- ◊ Session S2 - STARTED - Track 0 is A (extn 1000), Track 1 is B (extn 3000)
- ◊ Session S3 - STARTED - Track 0 is B (extn 3000), Track 1 is A (extn 1000)

**Notes:**

- ◊ Session S1 ends because Agent A's phone has placed Caller C on hold.
- ◊ Sessions S2 and S3 exist because both phones are configured for forking.
- ◊ The participants as well as the xRefCi for the two participants in S2 and S3 are identical but in reversed positions from each other.
- ◊ S1 xRefCi values are not reflected in either S2 or S3, since the consult is considered an independent call.

3. Agent A (extn 1000) completes conference.

C (extn 2000) talking to A (extn 1000) and B (extn 3000)

- ◊ Session S2 - ENDED
- ◊ Session S3 - ENDED
- ◊ Session S4 - STARTED - Track 0 is A (extn 1000), Track 1 is Conference Bridge
- ◊ Session S5 - STARTED - Track 0 is B (extn 3000), Track 1 is Conference Bridge

**Notes:** A far end transfer triggers the end of one recording session and the start of another recording. The completion of a conference is implemented as follows:

- ◊ During the consult:
  - · The consulting phone has a primary call on hold and an active consult call.
  - · The consulted phone only has one active call (the consult call)
- ◊ When the conference is completed (all parties connected):
  - · The consulting phone's consult call terminates.
  - · The consulting phone's primary call gets a far end transfer to the conference bridge.
  - · The consulted phone gets a far end transfer to the conference bridge.
- ◊ As a result:
  - · Two new sessions are created because both Agent A and Agent B have forking enabled.
  - · S4 near-end xRefCi value and S1 near-end xRefCi value will match.
  - · S5 near-end xRefCi value and S3 near-end xRefCi value will match.
  - · The far end xRefCi values for S4 and S5 will not match, even though both are connected to the same conference bridge.

4. Agent A (extn 1000) drops from conference.

A (extn 1000) disconnected. C (extn 2000) talking to B (extn 3000)

- ◊ Session S4 - ENDED
- ◊ Session S5 - ENDED
- ◊ Session S6 - STARTED - Track 0 is B (extn 3000), Track 1 is C (extn 2000)

**Notes:**

- ◊ The de-escalation of a conference into a normal two-party call is implemented as both remaining phones' far end transferring to each other.
- ◊ A far end transfer triggers the end of one recording session and the start of another recording session
- ◊ S6 and S5 will have matching near-end xRefCi values. Note that only one session remains active because Caller C does not have forking enabled.

5. Agent B (extn 3000) hangs up.

C (extn 2000) and B (extn 3000) disconnected.
Session S6 - ENDED

**Notes:**

- ◊ A far end transfer results in the end of one session and the start of another.
- ◊ A conference is created by transferring all phones to the conference bridge. Therefore a conference acts just like a set of transfers. Existing sessions are ended and new sessions are created between the forking phones and the conference bridge.
- ◊ Once the third party drops from the conference, the parties are transferred to each other. This ends the sessions which included the conference bridge and starts new sessions between the two remaining endpoints.
- ◊ If a fourth party is added to the conference bridge, there is no indication in the metadata, unless the fourth party also has its own forking enabled.

Metadata for sessions in case of Unified Communications Manager 10.x and later

# How do you correlate reference call IDs for different call sessions, under Unified Border Element forking?

With Unified Border Element forking, very few situations cause a call to be split into multiple recording sessions. Hold/Resume, transfer and conference operations do not start new recording sessions in most cases. In the few cases where new sessions are created, there is a common value, CCID (Call Correlation ID). This value is common to all sessions in the call. CCID is the decimal form of Cisco-GUID, a unique call key which is generated by Cisco voice routers. The first router which receives a call generates this key, and passes it down the line to all subsequent devices including Cisco MediaSense.

Unified Border Element itself does not generate xRefCi values, but to create similarity with Unified Communications Manager phone forking calls, Cisco MediaSense also synthesizes a pair of xRefCi values for every Unified Border Element call. These can be seen in the metadata at the track level, along with CCID, which appears at the session level.

The following situations cause Unified Border Element recordings to be split into multiple sessions.

## Mid-call Codec Change

If a transfer, conference, conference drop, or other operation causes the parties to renegotiate their codec, Cisco MediaSense ends the current recording session and starts a new one. The two sessions share the same CCID and the same pair of xRefCi values.

## Consult Transfers

A consult transfer is a transfer from one agent to another, in which the two agents talk to each other while the original caller waits on hold. The consult leg of the call is obviously related in some way to the overall call, and it is possible to configure Unified Communications Manager such that consult calls DO pass through CUBE. However, Unified Border Element and Cisco MediaSense do not know that these calls are related, and they create a new CCID and a new pair of xRefCi values for this session.

These calls can be associated with one another by comparing participant deviceRef and timestamp fields. Consider this scenario:

1. Caller C (extn 2000) calls Agent A (extn 1000) (sessionId = S1, CCID = C1)
2. Agent A consults with Agent B (extn 3000) (sessionId = S2, CCID = C2)
3. Agent A drops, and Caller C talks with Agent B (sessionId = S1, CCID = C1)

The red flag in this scenario is in Step 2. During that period, Agent A (deviceRef 1000) is a participant in two recording sessions at once:

- Session = S1 / CCID = C1 and
- Session = S2 / CCID = C2

Therefore, S1 is related to S2 and C1 is related to C2.

## Detecting Consult Calls

First we need a clear definition of consult call: *Any secondary call that is made by a current participant in an existing session to an endpoint which is outside that session and that excludes the other participants in that session.* In theory, this scenario could include an agent placing the caller on hold to check with his boss about taking lunch, or even an agent putting the caller on hold to receive a call from his wife, but we will ignore those possibilities for now.

It is possible for a client application to detect a consult call in real time by tracking the Cisco MediaSense event stream. If the client observes a session STARTED event containing a given deviceRef, followed by *another* session STARTED event containing the same deviceRef with no intervening session ENDED event, it can conclude that the sessionIds and the CCIDs found in the two session STARTED events are associated.

Historically, a client can check for any consult calls which are associated with a given primary call, using the Cisco MediaSense API. Assume the client knows that Agent A was using extn 1000, in CCID <C1>. It can follow these instructions to find any associated consult calls:

- Retrieve the session metadata for the primary call by issuing getSessionByCCID(<C1>).
- Extract the sessionStartDate (call it <Ta>), and sessionDuration.
- Calculate the sessionEndDate (call it <Tb>) by adding sessionDuration to <Ta>.
- Issue the following API request:

```
https://10.194.118.1:8443/ora/queryService/query/getSessionsByDeviceRef?value=1000&minSessi
```

This query could return more than one session. If it does, then all of them can be assumed to be associated with the same call.

## Detecting Consult Calls from Multiple Participants

The above procedure will find all consult calls made from the device which received the initial phone call. However, what if there are consult calls made from a device to which the call was subsequently transferred?

Consider this procedure:

- Caller calls Agent 1
- Agent 1 consults with Agent 2, then drops
- Caller speaks with Agent 2
- Agent 2 consults with Agent 3, then drops
- Caller speaks with Agent 3

The procedure above will not catch the consult call between Agent 2 and Agent 3.

Since this is a Unified Border Element call, we can make use of the fact that all of the connections between the caller and each of the agents are included in the same recording session, and the fact that *all* of the agents involved will be listed as participants in the same session at one time or another. Thus, from the primary session metadata, we can collect a list of all the deviceRefs which were involved. To find those sessions, we can make a series of calls to getSessionsByDeviceRef, specifying the primary session's time range, together with one deviceRef per request. Alternatively, we can shortcut that process by issuing a single getSessions request such as the following:

```
{
    "requestParameters": [
        {
            "fieldName": "deviceRef",
            "fieldConditions": [
                {
                    "fieldOperator": "equals",
                    "fieldValues": [
                        "1000"
                    ],
                    "fieldConnector": "OR"
                },
                {
                    "fieldOperator": "equals",
                    "fieldValues": [
                        "2000"
                    ],
                    "fieldConnector": "OR"
                },
                {
                    "fieldOperator": "equals",
                    "fieldValues": [
                        "3000"
                    ],
                    "fieldConnector": "OR"
                },
                {
                    "fieldOperator": "equals",
                    "fieldValues": [
                        "4000"
                    ]
                }
            ],
            "paramConnector": "AND"
        },
        {
            "fieldName": "sessionStartDate",
            "fieldConditions": [
                {
                    "fieldOperator": "between",
                    "fieldValues": [
                        <Ta>, // session start time
                        <Tb>// session end time
                    ]
                }
            ]
        }
    ]
}
```

This query will return all the consult calls associated with the original primary call and all of its transfers.

The astute reader will note that the above procedure actually casts the net too broadly. If for example, the agent at deviceRef 4000 conducted and terminated a completely independent call which happened to start *after* <Ta> and *before* he was added to the call in question, the above procedure will include that independent call in the set. This problem can be solved though, using the information available in the primary session's metadata. Each participant's information includes the time offset at which he joined the session and the duration of his tenure. Client code could use that information to simply delete the unrelated sessions from the list it received above. Or, it could formulate a series of direct getSessions or getSessionByDeviceRef queries which correctly frame the time periods during which each agent was in the primary call. We leave that as an exercise to the reader.

### Putting it All Together

In the preceding section, we presented precedures for retrieving all sessions associated with a given Cisco MediaSense recording session. However, we have also seen that a given call may be divided into more than one session, as in the case of a mid-call codec change.

How do we retrieve *all* recordings (including consults) associated with *all* sessions connected to the caller's interaction?

The answer is to extend the instructions above for detecting multiple consult calls. First, we would collect all sessions which share the CCID of the primary session in question. Then, we would build our list of participants from *all* of those session records. Next, we would calculate the time ranges as the sessionStartDate of the earliest session through the end of the latest session. Finally, we can execute the getSessions query shown above.

As before, we may end up capturing too many recordings, so the client could execute a postprocessing step to delete those unrelated sessions from its list.

## How do you associate calls in Cisco MediaSense with their appearances in other solution components?

### Identifier Correlation Table

Following are two tables?one for Unified Communications Manager calls and one for Unified Border Element calls. Each column represents a solution component or protocol, with the first column representing Cisco MediaSense. Each row represents a particular type of identifier.

To read the tables, begin with a cell that represents the data item that you know, and then look horizontally to the column representing the solution component in which you want to find the call. The entry in that cell indicates by what name the exact same data item is known in the target component. If the target component has a blank cell in that row, then that data item is not known to that component. You can instead look for an intervening column where you can cross vertically into another row where that cell is not blank in the target component's column.

For example, with a Unified Communications Manager call, assume that you know the GED-188 CallReferenceID and that you want to find the call in Cisco MediaSense. Looking left from the GED-188 column you, see that there is no value in the MediaSense column, so you cannot map directly to it. However, there is a column where you can zig-zag across rows: Unified Communications Manager CDR. A client can select the proper Unified Communications Manager CDR record by searching for one in which the IncomingProtocolCallRef matches the GED-188 CallReferenceID. That record contains a value called destLegCallIdentifier, which is the same as the

MediaSense NearEnd xRefCi, and can therefore be used to find the corresponding record in Cisco MediaSense.

Unified Communications Manager CDR records are not written until some time after the end of the call terminates, however, so the above method can only be used historically.

There is another path as well. Look downward from the GED-188 CallReferenceID. It turns out that you can also use the AlertingDevice and AnsweringDevice to match the deviceRef field in MediaSense. This method also works in real time.

**Call Correlation for Calls Forked by a Unified CM IP Phone**

| MediaSense | Ingress Gateway or CUBE | AAA RADIUS CDR | UCM CDR | TAPI/JTAPI field | U |
|---|---|---|---|---|---|
| (1) | Cisco-GUID | | IncomingProtocolCallRef | CiscoConnection.UniqueID | TCD.Ca |
| NearEnd xRefCi | | | destLegCallIdentifier | Terminal.ConnectionID | |
| FarEnd xRefCi | | | origLegCallIdentifier | Terminal.ConnectionID | |
| | | | global_CallID_call-ManagerId + global_CallID_callId (A.K.A. UCM GCID) | CiscoCall.CallID | TCD.Pe |
| deviceId | | | | | |
| deviceRef | | | | | TCD.Ins (2) |

**Call Correlation for Calls Forked by CUBE**

| MediaSense | Ingress Gateway or CUBE | AAA RADIUS CDR | UCM CDR | TAPI/JTAPI field | UCCE Database |
|---|---|---|---|---|---|
| CCID (3) | Cisco-GUID | Cisco-GUID | IncomingProtocolCallRef | | TCD.CallGUID |
| deviceRef | Called or calling party extn | | | | TCD.InstrumentPortNu (2) |

## Notes

1. In recordings of Unified Communications Manager calls, Cisco MediaSense does in fact receive a Cisco-GUID from UCM, but it is not the same one that is captured by the other solution devices. MediaSense therefore does not even store this value.

2. For agent-to-agent calls, TCD.InstrumentPortNumber is the destination agent?s extension. The calling agent?s extension can be found in TCD.ANI.
3. CCID is the Cisco-GUID in decimal form, which is 4 hyphen-separated sets of 10-digit decimal numbers. These can be converted to the hex form by simply converting each 10-digit decimal number to an 8-digit hex number, and removing the hyphens. Where the Cisco-GUID is used in UCCE, it is in its hex form.

# How do you determine which track has the calling party and which track has the called party?

## For Calls Forked by CUBE

For CUBE calls, Track 0 always maps to the Anchor leg media stream. The Anchor leg is the dial-peer which media recording profile is configured. The second track maps to non-anchor leg.

If you have media recording profile enabled on the *inbound* dialpeer, then the anchor leg becomes the in-leg. In other words, the calling party appears in Track 0, and the called party appears in Track 1.

If you have media recording profile enabled on the *outbound* dialpeer, then the anchor leg becomes the out-leg.  In that case the calling party appears in Track 1 and the called party appears in Track 0.

## For Calls Forked by Unified CM Phones

For Unified CM forking, in simple call scenarios you can use the xRefCi fields in the metadata to determine which party is in which media track. The numerically smaller xRefCi usually refers to the calling party's track. The called party's track will be numerically larger (usually by one, but it could be more under a reasonably loaded system). However, these xRefCi values eventually wrap around to zero. So, if you find that one value is a high number and the other is a small number, you should assume their positions are reversed.

In more complicated scenarios, this algorithm does not always work. If supplementary services are invoked, such as transfers and conferences, and the UC Manager cluster consists of more than one node, then the xRefCi values are not necessarily generated sequentially, and you cannot assume that their order has any meaning at all. A direct way to determine whether the ordering sequence of a particular pair of xRefCi values can be trusted is to look at the first byte of the xRefCi values. This byte represents the UC Manager Node ID on which that particular identifier was created. If the first bytes of the two xRefCi values are the same, then their ordering is correct. If they are different, then the ordering might not be correct.

For these cases, the only way to determine call direction in real time is to acquire the information from any other source, such as the JTAPI event feed. Once the call has ended and a few minutes have elapsed, you can always determine the call direction by examining UC Manager's CDR data for the call. Specifically, the origLegCallIdentifier field in the CDR record will always represent the caller.

# What are the possible causes for a session state of CLOSED_ERROR?

Possible causes for a session state of CLOSED_ERROR include:

1. Call control server got an error response from the Media (recording) server for the open or close request
2. Call control server detected a SIP signaling error, for example a missing ACK
3. Session was successfully closed, but ALL tracks have zero size

When a session is in the ACTIVE state, it is normal that there is no duration in the metadata, because duration is not known until session is closed.

For a session that is in CLOSED_ERROR state, if either the session or track duration fields are not present in either the event or the getSessions data, then media for this track is not available.

# What is the difference between Pruned and Deleted sessions?

Consider the following two queries:

## Query 1: Using getAllPrunedSessions

```
https://10.192.252.171:8443/ora/queryService/query/getAllPrunedSessions?minSessionStartDate
```

This query returns a set of sessions, all of whose session states are DELETED.

## Query 2: Using getSessions

```
https://10.192.252.171:8443/ora/queryService/query/getSessions
{
            "requestParameters":
            [{
                        "fieldName" : "sessionState",
                        "fieldConditions":
                        [{
                                    "fieldOperator" : "equals",
                                    "fieldValues" : [ "DELETED" ]
                        }],
                        "paramConnector" : "AND"
            },
            {
                        "fieldName" : "sessionStartDate",
                        "fieldConditions":
                        [{
                                    "fieldOperator" : "between",
                                    "fieldValues" : [ "1301788800000", "1312329599000" ]
                        }]
            }]
}
```

This query returns no sessions.

## Why the behavior difference in pruned and deleted sessions?

The behavior difference is by design. Refer to the following sections in the MediaSense Documentation:

- The API Parameter Description: The getAllPrunedSessions API description: "Use this API to search all pruned recordings...The term Pruned refers to recordings which are deleted by the Cisco MediaSense system. If you have explicitly deleted any recording using the deleteSessions API, then these deleted recordings are not considered as pruned recordings."

- The MediaSense SRND under the section "Proactive Storage Management": "When sessions are pruned, the metadata that is associated with these sessions remains in the database, even after these sessions are marked as 'pruned'. This metadata does not take a large amount of storage space compared to the recordings themselves but it does take some space and should be periodically removed.

  To aid in this activity, clients may periodically issue an API request for pruned sessions, or clients may elect to receive session pruned events and explicitly delete those events that clients no longer need."

To clarify, the two queries above are entirely different. As a matter of fact, the second query (looking for all sessions whose state is "DELETED") will ALWAYS return an empty set. Normal day-to-day queries filter out sessions with DELETED states, even if that is what is being requested. The only exception is "getAllPrunedSessions". This exception is intended to assist the application in finding pruned sessions so that the application can request that these sessions be deleted.

Once you use the "deleteSessions" API on the list of pruned sessions you get from "getAllPrunedSessions", these sessions will NO LONGER appear in the result of "getAllPrunedSessions". Such sessions will be completely removed from the metadata immediately.

Another way to look at this is that "pruned" sessions are not the same thing as "deleted" sessions:

- Pruned sessions have been marked for removal by an algorithm in the MediaSense system. No person was involved in the decision to prune these sessions. So even though these sessions are moved to the "DELETED" state, these sessions are NOT actually removed from the metadata. Human (or Application) intervention is required. Because these sessions are in the "DELETED" state, these sessions are not visible to most queries. However, these sessions ARE visible to the "getAllPrunedSessions" query API. Also, if any mp4 files were generated for these sessions, these mp4 files continue to be present on the disk and continue to be available for download until the pruned sessions are actually DELETED.
- Deleted sessions are marked by explicitly calling the "deleteSessions" API. This marking can be done to sessions that are already pruned or to sessions that have not yet been deleted. Once a session has been deleted by the deleteSessions API, this session is no longer visible to ANY query. This includes the getAllPrunedSessions API. These deleted sessions are removed from the metadata immediately so that disk space can be reclaimed.

# How to Configure a TDM Gateway for Media Forking?

The basic problem is this: you have a PSTN gateway through which calls are flowing, and you want to record those calls. These calls are TDM-to-SIP calls. However, media forking is only available on SIP-to-SIP calls.

It turns out that you can indeed record these calls, by directing them through the router a second time. Configuration guidance and other details can be found in this white paper.

# How to Capture Actual Destination Phone When Using a Hunt Group?

When you are forking media from CUBE, the MediaSense metadata normally contains the extension of the called party. However, if the number called is a Communications Manager hunt group pilot number, then by default the metadata will only contain that pilot number. It will not contain the extension of the phone which actually answered the call.

There is a Communications Manager setting which can change this. On the Hunt/Pilot Configuration Page, find the section entitled "Connected Party Transformations". The setting "Display Line Group Member DN as Connected Party" should be turned on.

This capability is available as of Communications Manager 9.0(1).

# Why Unified Communications Manager Network-Based Recording is recommended as a preferred forking mechanism?

With Unified Communications Manager Network-based Recording (NBR), you can use a gateway to record calls. NBR allows the Unified Communications Manager to route recording calls, regardless of device, location, or geography. With NBR, call recording media can be sourced from either the IP phone or from a gateway that is connected to the Unified Communications Manager over a SIP trunk. Unified Communications Manager dynamically selects the right media source based on the call flow and call participants.

NBR offers an automatic fallback to BiB when the Integrated Services Routers (ISR) are unavailable as no separate recording configuration is required. This is useful in cases where customers want to include agent-agent consult calls in the recording policies as Unified Border Element cannot record consult calls, so BiB needs to be enabled separately.

Both NBR and BiB calls can be correlated using xRefci, which is available from Unified Communications Manager JTAPI; CISCO-GUID is not needed, which means neither CTI server nor CTIOS connections are required. As there is a single correlation identifier, correlation across components is stronger and can be done in a uniform way independent of the call flow.

Using NBR, directly-dialed as well as dialer-initiated outbound calls can be correlated with their appearance in other solution components.

Using NBR, TDM gateway recording is automatically used without splitting the capacity of the router. Currently, TDM gateway recording is not supported with MediaSense 10.5.

# Why a node takes longer to upgrade to MediaSense 10.5?

A node can take several hours to upgrade depending on the number and size of recordings it holds. For MediaSense 10.5, when you upgrade a node with very large data sets, it takes around 90 additional minutes per 1 million recordings.

# What is the impact of the Russian time zone changes on the MediaSense Search and Play application?

Users of the MediaSense Search and Play application will be affected if they are located in any of the impacted time zones or if they select an impacted time zone in the search criteria. The third party partner products which interface with MediaSense will be affected similarly until they update their respective time zone tables.

The workaround is to select a time zone that matches the correct offset from GMT even if the city is no longer correct.

# What are the languages supported by MediaSense?

The following languages are supported by MediaSense:

- Arabic
- Danish
- Dutch
- English (United States)

- Finnish
- French
- German
- Italian
- Japanese
- Korean
- Norwegian
- Polish
- Portuguese (Brazilian)
- Russian
- Simplified Chinese
- Spanish
- Swedish
- Traditional Chinese
- Turkish

# How to monitor MediaSense system performance?

To monitor MediaSense system performance, analyze the values of the following key performance indicators (KPIs) using the RTMT tool or Cisco Prime Collaboration Assurance tool.

For more information on RTMT tool or Cisco Prime Collaboration Assurance tool, refer the "Unified RTMT Administration" and "Cisco Prime Collaboration Assurance Administration" sections of Cisco MediaSense User Guide.

**KPIs and their Threshold Values**

| Key Performance Indicator | RTMT Counters | Suggested Threshold Values |
|---|---|---|
| Call success rate | MediaSense Call Control Service > Number of recording sessions without errors<br><br>MediaSense Call Control Service > Number of recording sessions with errors | 99.99%<br><br>Call Success rate = number of recording sessions without errors / (number of recording sessions without errors + number of recording sessions with errors) * 100 |
| APIs response mean time for API | Cisco MediaSense API Service > Mean query response time | 60 secs |
| Recording start mean setup delay | Cisco MediaSense Call Control Service > Mean setup delay | 3 secs |
| CPU Mean Utilization | Processor > %CPU Time | 90% |
| Memory Mean Utilization | Memory > %Mem Used | 70% |
| | | |

| | | |
|---|---|---|
| RTP/UDP packet drop | Network Interface > Rx Dropped > eth0<br><br>Network Interface > Rx Errors > eth0 | 0<br><br>0 |

# How to configure a browser to run in-browser player in MediaSense?

Based on the browser, perform the following steps to run in-browser player.

| Internet Explorer 9 | Internet Explorer 11 | Mozilla Firefox |
|---|---|---|
| 1. In **MediaSense Search and Play**, click the **Play** icon of a recording session. | Pre-requisites: In case of fresh install of MediaSense 11.0, ensure that MediaSense nodes are added to the cluster using the respective Fully Qualified Domain Name (FQDN).<br><br>In case of an upgrade to MediaSense 11.0, ensure that the MediaSense nodes that were previously added using the hostname, should now be displayed by the respective FQDN. Check the MediaSense Server List in **MediaSense Server Configuration** window (**Cisco MediaSense Administration > System > MediaSense Server Configuration**). | 1. Add the self-signed certificate of a MediaSense node for port 8446 of mp4url in Mozilla Firefox's trusted authority. |
| 2. Click **Yes** to trust the certificate.<br><br>**Note:** Verify that the self-signed certificate offered is of the targeted MediaSense node by validating the FQDN in the certificate?s technical details. | Perform the following steps:<br><br>1. Set the *set hostnameformediaurl* CLI as "true" to make MediaSense prepare the mp4url and rest of the mediaurls using FQDN only.<br><br>`admin:set useHostNameForMediaURL`<br>`admin:set useHostNameForMediaURL true`<br><br>2. Restart the Configuration Service to activate the property.<br><br>`admin:utils service restart Cisco MediaSense Configuration Service`<br><br>**Note:** If the service has not restarted properly, execute the same command again.<br><br>3. After the Configuration Service restarts, sign out and sign in to **MediaSense Search and Play**.<br><br>**Limitation:** In case the MediaSense nodes were previously added using the IPs, then the nodes continue to be displayed by the IPs only even after an upgrade to MediaSense 11.0. In-browser player does not work on Internet Explorer 11, irrespective of the *hostnameformediaurl* CLI command's value. In this case, it is recommended that the *hostnameformediaurl* CLI command should not be set as "true." | 2. To add the self-signed certificate, click the **download** icon of a recording session and select **mp4**.<br><br>This **Connection is Untrusted** pop-up window appears. |

| | | |
|---|---|---|
| 3. Click the **Play** icon corresponding to the selected recording.<br><br>The in-browser player plays the selected recording session. | Perform the following steps to add the MediaSense self-signed certificate to Windows trusted authority.<br><br>1. Open **MediaSense Search and Play**.<br>A security certificate pop-up window appears.<br>2. Click **Continue**.<br>**MediaSense Search and Play** window appears.<br>3. In the Address bar, click the **Certificate Error** icon.<br>4. Click **View certificates**.<br>The certificate pop-up window appears.<br>5. Click **Install Certificate**.<br>The Certificate Import Wizard appears.<br>6. Click **Next**.<br>7. In the **Certificate store** window, select the **Place all certificates in the following store** radio button and click **Browse**.<br>The **Select Certificate Store** dialog box appears.<br>8. Check the **Show physical stores** check box and select the **Trusted Root Certificationhorities** folder.<br>9. Click **OK** and **Next**.<br>10. Click **Finish** to complete the certificate import.<br>A security warning pop-up window appears to confirm the installation of the certificate.<br>11. Click **Yes**.<br>The following message appears.<br><br>`The import was successful.`<br><br>12. Click **OK**.<br>13. Click **OK** on the **Certificate** pop-up window.<br>14. Close and open the browser.<br>15. Open **MediaSense Search and Play**.<br>The security certificate issxvi still persists.<br>16. Go to **Tools > Internet Options > Advanced**, uncheck the **Warn about certificate address mismatch** check boxviiunder Security.<br>17. Click **Apply** and **OK**.<br>18. Restart the browser and open **MediaSense Search and Play**.<br><br>Ensure that the MediaSense server is accessible through FQDN on the browser. If not, navigate to C:\Windows\System32\drivers\etc, open the **hosts** file in Notepad and add the IP address of MediaSense server and its FQDN at the bottom of the file. In-browser player starts working on Internet Explorer 11.<br><br>**Note:** If a recording is present on a different MediaSense node of the cluster, you are prompted to add the certificate of that MediaSense node in the trusted authority. | 2. To add the self-signed certificate, click the **download** icon of a recording session and select **mp4**.<br><br>**This Connection is Untrusted** pop-up window appears.<br>**Note:** Verify that the self-signed certificate offered is of the targeted MediaSense node by validating the FQDN in the certificate?s technical details. |
| | | 3. Click **I Understand the Risks** link. |
| | | 4. Click **Add Exception**. |

| | | |
|---|---|---|
| | | The **Add Security Exception** pop-up window appears. |
| | | 5. Click **Confirm Security Exception**.<br><br>The self-signed certificate of the particular MS node of 8446 port gets added to the browser's trusted authority. |