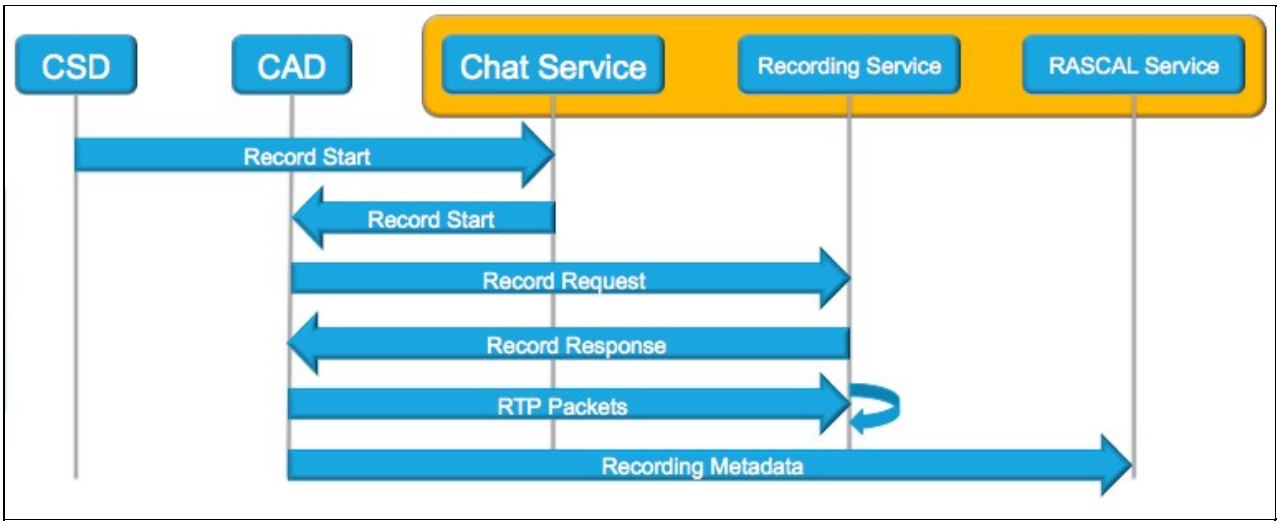
Contents

CAD Desktop Recording Log Analysis

Purpose

This document serves as a sample log analysis from a working UCCX CAD Desktop Recording Session. The goal is to increase knowledge in the field of troubleshooting CAD Desktop Recording by providing good, working examples to compare to production environments.

Communication Flow Diagram



Log Information

When tracing communication through CAD Desktop Recording logs, use the following logs. In most cases DEBUG level tracing is sufficient except where specified in the following log snippets. When using TRACE level settings, be sure to consult Cisco TAC. TRACE level may have a negative impact on performance and should not be enable during high production hours.

When retrieving CCX Service logs, collect them from the service's active side, with exception of the Recording Service. The Recording Service uses a round robin approach to picking the CCX server to service the CAD Desktop Recording request. The CAD agent.dbg log will specify which recording service it is using on which CCX server (see below).

- Agent.dbg
  - C:\Program Files\Cisco\Desktop\log on Agent?s PC
- Supervisor.dbg

- ♦ C:\Program Files\Cisco\Desktop\log on Supervisor?s PC
- FCCServer.dbg
  - ♦ Collected via RTMT (Call/Chat Service)
- RPServer.dbg
  - ♦ Collected via RTMT (Recording Service)
- FCRasSrv.dbg
  - ♦ Collected via RTMT (Recording and Statistics Service)

Keywords

When tracing communication through CAD Desktop Recording logs, the following keywords serve as unique identifiers for transactions.

- Agent Extension
- ConnectionCallID
- Recording ID
- Message Type
- IP Addresses

Log Communication Analysis

Start with the **`CALL_ESTABLISHED_EVENT`** in the Agent.dbg log to find the **`ConnectionCallID`**. Ensure that **`IsBusinessCall`** is **`true`** otherwise the agent is not considered applicable for Desktop Recording.

`Agent.dbg`

```
2012-03-22 16:04:38:024 DEBUG [0x468] EventHandler.cpp[2141] GetDebugInfo:
 GetDebugInfo ---------------------- Begin: CALL_ESTABLISHED_EVENT ----------------------
2012-03-22 16:04:38:024 DEBUG [0x468] EventHandler.cpp[2263] GetDebugInfo: CALL_ESTABLISHED_EVENT:
2012-03-22 16:04:38:024 DEBUG [0x468] EventHandler.cpp[2264] GetDebugInfo: CALL_ESTABLISHED_EVENT:
2012-03-22 16:04:38:025 DEBUG [0x468] EventHandler.cpp[2265] GetDebugInfo: CALL_ESTABLISHED_EVENT:
2012-03-22 16:04:38:025 DEBUG [0x468] EventHandler.cpp[2266] GetDebugInfo: CALL_ESTABLISHED_EVENT:
2012-03-22 16:04:38:025 DEBUG [0x468] EventHandler.cpp[2267] GetDebugInfo: CALL_ESTABLISHED_EVENT:
2012-03-22 16:04:38:025 DEBUG [0x468] EventHandler.cpp[2268] GetDebugInfo: CALL_ESTABLISHED_EVENT:
2012-03-22 16:04:38:025 DEBUG [0x468] EventHandler.cpp[2269] GetDebugInfo: CALL_ESTABLISHED_EVENT:
2012-03-22 16:04:38:025 DEBUG [0x468] EventHandler.cpp[2270] GetDebugInfo: CALL_ESTABLISHED_EVENT:
2012-03-22 16:04:38:027 DEBUG [0x468] EventHandler.cpp[2271] GetDebugInfo: CALL_ESTABLISHED_EVENT:
2012-03-22 16:04:38:027 DEBUG [0x468] EventHandler.cpp[2272] GetDebugInfo: CALL_ESTABLISHED_EVENT:
2012-03-22 16:04:38:027 DEBUG [0x468] EventHandler.cpp[2273] GetDebugInfo: CALL_ESTABLISHED_EVENT:
2012-03-22 16:04:38:027 DEBUG [0x468] EventHandler.cpp[2458] GetDebugInfo:
 GetDebugInfo ---------------------- End: CALL_ESTABLISHED_EVENT ----------------------
```

The following entry will be printed repeatedly until both **`address1`** and **`address2`** fields are filled in, indicating the IP addresses of the incoming and outgoing RTP streams. Note that **`callCanBeMonitored`** will return **`false`** until such IP addresses are obtained.

`Agent.dbg`

```
2012-03-22 16:04:38:030 DEBUG [0x468] AgentMonitor.cpp[539] AgentMonitor::callCanBeMonitored:
 isTheActiveCall true, isABusinessCall true, forwardOnlyBusinessCalls true, address1 [], address2
2012-03-22 16:04:38:030 TRACE [0x468] AgentMonitor.cpp[551] AgentMonitor::callCanBeMonitored:
 Call can't be monitored because one or both rtp stream info has blank ip address, probably waitin
2012-03-22 16:04:38:030 DEBUG [0x468] AgentMonitor.cpp[556] AgentMonitor::callCanBeMonitored: Retu
```

Look for the **`OnRTPStartedEvent`** for both the out going (**`RTP Direction OUTPUT`**) and in coming

(**RTP Direction INPUT**) streams. CAD uses this events to determine the IP addresses needed to capture RTP traffic.

Agent.dbg

```
2012-03-22 16:04:38:925 DEBUG [0x468] EventHandler.cpp[1977] OnRtpStartEvent: ========== OnRTPStar
2012-03-22 16:04:38:925 DEBUG [0x468] EventHandler.cpp[1978] OnRtpStartEvent: OnRTPStartedEvent: C
2012-03-22 16:04:38:925 DEBUG [0x468] EventHandler.cpp[1982] OnRtpStartEvent: OnRTPStartedEvent: C
2012-03-22 16:04:38:925 DEBUG [0x468] EventHandler.cpp[1983] OnRtpStartEvent: OnRTPStartedEvent: C
2012-03-22 16:04:38:925 DEBUG [0x468] EventHandler.cpp[1984] OnRtpStartEvent:
 OnRTPStartedEvent: Direction of Audio Stream: RTP Direction OUTPUT
2012-03-22 16:04:38:925 DEBUG [0x468] EventHandler.cpp[1991] OnRtpStartEvent: ========== End OnRTP
```

Agent.dbg

```
2012-03-22 16:04:38:937 DEBUG [0x468] EventHandler.cpp[1977] OnRtpStartEvent: ========== OnRTPStar
2012-03-22 16:04:38:937 DEBUG [0x468] EventHandler.cpp[1978] OnRtpStartEvent: OnRTPStartedEvent: C
2012-03-22 16:04:38:937 DEBUG [0x468] EventHandler.cpp[1982] OnRtpStartEvent: OnRTPStartedEvent: C
2012-03-22 16:04:38:937 DEBUG [0x468] EventHandler.cpp[1983] OnRtpStartEvent: OnRTPStartedEvent: C
2012-03-22 16:04:38:937 DEBUG [0x468] EventHandler.cpp[1984] OnRtpStartEvent:
 OnRTPStartedEvent: Direction of Audio Stream: RTP Direction INPUT
2012-03-22 16:04:38:937 DEBUG [0x468] EventHandler.cpp[1991] OnRtpStartEvent: ========== End OnRTP
```

Having both IP addresses, CAD will report **callCanBeMonitored** as **true**.

Agent.dbg

```
2012-03-22 16:04:38:942 DEBUG [0x468] AgentMonitor.cpp[539] AgentMonitor::callCanBeMonitored:
 isTheActiveCall true, isABusinessCall true, forwardOnlyBusinessCalls true, address1 [10.77.85.104
2012-03-22 16:04:38:942 DEBUG [0x468] AgentMonitor.cpp[556] AgentMonitor::callCanBeMonitored: Retu
```

In the case where the supervisor initiated CAD Desktop Recording through CSD, we will see an entry in the Supervisor.dbg log showing that the supervisor clicked the start record button.

Supervisor.dbg

```
2012-03-22 16:04:11:618 CALL [0xaa4] AgentView.cpp[3539] CAgentView::OnSupervisorRecord: AV3539 Us
2012-03-22 16:04:11:621 DEBUG [0xaa4] AgentView.cpp[2943] CAgentView::GetSelectedCallID: AV2943 Ge
```

CSD will create a **genericMessage** to send to CAD, through the Chat Service. The **ConnectionCallID** will be stored in the **dataLongMap[0]** field.

Supervisor.dbg

```
2012-03-22 16:04:11:692 CALL [0xaa4] MonitoringAndRecording.cpp[152] MonitoringAndRecording::sendM
2012-03-22 16:04:11:692 CALL [0xaa4] MonitoringAndRecording.cpp[161] MonitoringAndRecording::sendM
 AV0161 Requesting recording to start at agent 85104 for callID 16778275
2012-03-22 16:04:11:692 DEBUG [0xaa4] FCCClientAPI.cpp[1609] FCCClientAPI::genericMessage: Begin.
2012-03-22 16:04:11:692 DEBUG [0xaa4] FCCClientAPI.cpp[1623] FCCClientAPI::genericMessage:
 Begin. source Ext: 1000, sourceType: Supervisor, destinationType: To Agent, destinationId: 85104,
2012-03-22 16:04:11:692 DEBUG [0xaa4] FCCClientAPI.cpp[1631] FCCClientAPI::genericMessage:   dataS
2012-03-22 16:04:11:692 DEBUG [0xaa4] FCCClientAPI.cpp[1642] FCCClientAPI::genericMessage:   dataI
2012-03-22 16:04:11:692 DEBUG [0xaa4] FCCClientAPI.cpp[1642] FCCClientAPI::genericMessage:   dataI
2012-03-22 16:04:11:692 DEBUG [0xaa4] FCCClientAPI.cpp[1679] FCCClientAPI::genericMessage: Calling
2012-03-22 16:04:11:694 DEBUG [0xaa4] FCCClientAPI.cpp[1730] FCCClientAPI::genericMessage: End.  r
2012-03-22 16:04:11:696 CALL [0xaa4] MonitoringAndRecording.cpp[152] MonitoringAndRecording::sendM
```

When tracing **genericMessage** events, note the **sourceType**, **destinationType**, and **messageType**. These fields indicate the direction of the message and which messages match each other in the various logs. Next, the Chat Service will receive the message, as indicated by these three fields, to pass to

CAD. The **ConnectionCallID** will be stored in the **ulongArgs[0]** field.

FCCServer.dbg

```
2012-03-22 16:04:32:622 DEBUG [0x1e83ba0] CChatServer.cpp[3179] genericMessage:
 Begin.  senderUserType: Supervisor, senderExtension: 1000, destType: To Agent, destinationId: 851
2012-03-22 16:04:32:622 DEBUG [0x1e83ba0] CChatServer.cpp[3189] genericMessage: <Supervisor_1000>
2012-03-22 16:04:32:622 DEBUG [0x1e83ba0] CChatServer.cpp[3220] genericMessage: Sender is a superv
2012-03-22 16:04:32:622 DEBUG [0x1e83ba0] CChatServer.cpp[3248] genericMessage: find agent.
2012-03-22 16:04:32:622 DEBUG [0x1e83ba0] CChatServer.cpp[3300] genericMessage: stringArgIndexes[0
2012-03-22 16:04:32:622 DEBUG [0x1e83ba0] CChatServer.cpp[3312] genericMessage: stringArgs[0] = 10
2012-03-22 16:04:32:622 DEBUG [0x1e83ba0] CChatServer.cpp[3323] genericMessage: ulongArgIndexes[0]
2012-03-22 16:04:32:622 DEBUG [0x1e83ba0] CChatServer.cpp[3323] genericMessage: ulongArgIndexes[1]
2012-03-22 16:04:32:622 DEBUG [0x1e83ba0] CChatServer.cpp[3333] genericMessage: ulongArgs[0] = 167
2012-03-22 16:04:32:623 DEBUG [0x1e83ba0] CChatServer.cpp[3333] genericMessage: ulongArgs[1] = 0.
```

Next CAD will receive this **genericMessage** from the Chat Service. This is matched up by the **sourceType**, **destinationType**, and **messageType**. The **ConnectionCallID** is stored in the **genericMessage.dataStringMap[0]** field.

Agent.dbg

```
2012-03-22 16:04:54:254 DEBUG [0x4fc] FCC_Client_impl.cpp[800] FCC_Client_impl::genericMessage:
 Begin. destType: Agent, destID: 85104, senderUserType: Supervisor, senderId: 1000, messageType: 2
2012-03-22 16:04:54:254 DEBUG [0x4fc] FCC_Client_impl.cpp[804] FCC_Client_impl::genericMessage: st
2012-03-22 16:04:54:255 DEBUG [0x4fc] FCC_Client_impl.cpp[809] FCC_Client_impl::genericMessage: ul
2012-03-22 16:04:54:255 DEBUG [0x4fc] FCC_Client_impl.cpp[809] FCC_Client_impl::genericMessage: ul
2012-03-22 16:04:54:255 DEBUG [0x4fc] FCC_Client_impl.cpp[814] FCC_Client_impl::genericMessage: ul
2012-03-22 16:04:54:257 DEBUG [0x4fc] FCC_Client_impl.cpp[814] FCC_Client_impl::genericMessage: ul
2012-03-22 16:04:54:257 DEBUG [0x4fc] FCC_Client_impl.cpp[843] FCC_Client_impl::genericMessage: ge
2012-03-22 16:04:54:258 DEBUG [0x4fc] FCC_Client_impl.cpp[851] FCC_Client_impl::genericMessage: ge
2012-03-22 16:04:54:258 DEBUG [0x4fc] FCC_Client_impl.cpp[851] FCC_Client_impl::genericMessage: ge
```

CAD will initiate a license checkout for the agent.

Agent.dbg

```
2012-03-22 16:04:54:626 DEBUG [0x468] AbstractLRMClient.cpp[48] lrm::AbstractLRMClient::checkoutLi
 Checkout license <recording> succeeded.
```

With a successful Recording license checked out, CAD attempts to create the recording session. At this time a unique **recording.Session id** is created that will identify this recording request through its lifetime. Important to note the IP address being returned by **CRecordingClient::GetServer: Returning server:** as this indicates which UCCX server the recoding is being saved onto. The **ServerHost**, **portTo**, and **portFrom** fields will remain **0** until values are returned by the Recording Service on the UCCX server.

Agent.dbg

```
2012-03-22 16:04:54:641 DEBUG [0x468] FCRasClient.cpp[2214] FCRasClientBase::StartRecording2:
 Starting recording for Agent (agent2) on extension (85104).
2012-03-22 16:04:54:643 DEBUG [0x468] FCRasClient.cpp[2226] FCRasClientBase::StartRecording2:
 Starting the recording:(201203221604540000021851041000)!
2012-03-22 16:04:54:645 CALL [0x468] RecordingAPI.cpp[57] rpsRecordingStart:
 Begin. fileName = <201203221604540000021851041000>, serverId = <>, address = <0>, portTo = <0>, po
2012-03-22 16:04:54:646 CALL [0x468] CRecordingClient.cpp[455] CRecordingClient::Start: Begin.
2012-03-22 16:04:54:650 DEBUG [0x468] CRecordingClient.cpp[431] CRecordingClient::GetServer: There
2012-03-22 16:04:54:651 DEBUG [0x468] CRecordingClient.cpp[432] CRecordingClient::GetServer: There
2012-03-22 16:04:54:651 DEBUG [0x468] CRecordingClient.cpp[433] CRecordingClient::GetServer:
 The server with the order number = 0 will be chosen.
2012-03-22 16:04:54:652 DEBUG [0x468] CRecordingClient.cpp[446] CRecordingClient::GetServer: Retur
```

```
2012-03-22 16:04:54:652 DEBUG [0x468] CRecordingClient.cpp[472] CRecordingClient::Start: Trying se
2012-03-22 16:04:54:655 DEBUG [0x468] CRecordingClient.cpp[485] CRecordingClient::Start:
 Trying to start recording.Session id= <20120322160454000021851041000> ServerHost = <0>, portTo =
```

The Reporting Service will begin a series of internal checks. It will check the recording session id, capacity, then begin to allocate ports for the **portTo** and **portFrom** entities. Lastly, the Recording Service will create a .Raw file for the RTP packets to be saved.

RPServer.dbg

```
2012-03-22 16:04:33:091 CALL [0x1c30ba0] CRPSCritSection.cpp[802] RecordingStart: Begin.
2012-03-22 16:04:33:091 DEBUG [0x1c30ba0] CRPSCritSection.cpp[811] RecordingStart:
 Checking whether there's already a recording session with the same id.
2012-03-22 16:04:33:091 DEBUG [0x1c30ba0] CRPSCritSection.cpp[820] RecordingStart: This is a new i
2012-03-22 16:04:33:091 DEBUG [0x1c30ba0] CRPSCritSection.cpp[822] RecordingStart:
 Capacity is limited to 80 recordings, 99999 points, free space must be over 2048 MB.
2012-03-22 16:04:33:091 DEBUG [0x1c30ba0] CRPSCritSection.cpp[823] RecordingStart: There are curre
2012-03-22 16:04:33:092 DEBUG [0x1c30ba0] CRPSCritSection.cpp[877] RecordingStart: Acquiring ports
2012-03-22 16:04:33:092 DEBUG [0x1c30ba0] CRPSCritSection.cpp[885] RecordingStart: Got ports: To:(
2012-03-22 16:04:33:092 DEBUG [0x1c30ba0] CRPSCritSection.cpp[889] RecordingStart: Starting the re
2012-03-22 16:04:33:092 CALL [0x1c30ba0] CRecRawThread.cpp[81] Start:
 ENTRY p_sRTPFile=(/common/cisco/uccx/rshell/recording/20120322160454000021851041000), p_nClientTo
2012-03-22 16:04:33:093 DEBUG [0x2b0aba0] CRecRawThread.cpp[265] run:
 Saving to file:(/common/cisco/uccx/rshell/recording/20120322160454000021851041000.From.Raw).
```

Once CAD receives the response from the Recording Service, **ServerHost**, **portTo**, and **portFrom** will all have values.

Agent.dbg

```
2012-03-22 16:04:54:702 DEBUG [0x468] CRecordingClient.cpp[487] CRecordingClient::Start: Finished
 return code = <0>. sessionId = <20120322160454000021851041000>, nServerHost = <1683311882>, portT
2012-03-22 16:04:54:706 DEBUG [0x468] CRecordingClient.cpp[497] CRecordingClient::Start:
 Successfully started the recording (20120322160454000021851041000) on server (10.77.85.100).
```

CAD will identify the two RTP streams, indicated by **Stream ONE** and **Stream TWO**, to begin to capture the RTP traffic. The IP addresses were obtained from the **OnRTPStartedEvent** mentioned previously.

Agent.dbg

```
2012-03-22 16:04:54:775 TRACE [0x468] MonitoringBase.cpp[166] MonitoringBase::startMonitoringOrRec
 Stream ONE: AgentExt<85104> callid<0>, port<24584>, action<1920992373>, IPAddr<10.77.85.104>
2012-03-22 16:04:54:777 TRACE [0x468] MonitoringBase.cpp[167] MonitoringBase::startMonitoringOrRec
 Stream TWO: AgentExt<85104> callid<0>, port<24606>, action<108977880>, IPAddr<10.77.85.103>
```

Next CAD attempts to open the NIC identified for CAD Desktop Recoding in postinstall.exe. The logs will display which NIC adapter and the filter being used to capture RTP packets.

Agent.dbg

```
2012-03-22 16:04:55:937 INFO [0x884] VOIP2042 _snifferSessionThread: The NIC adapter was opened su
2012-03-22 16:04:55:939 TRACE [0x884] SnifferSession.cpp[191] SnifferSession::PacketFilter_lookupn
 Calling splk_pcap_lookupnet() with \Device\Splkpc_{671D2F1B-F54F-431B-8603-B17E8973C423}.
2012-03-22 16:04:55:958 TRACE [0x884] SnifferSession.cpp[208] SnifferSession::PacketFilter_lookupn
 splk_pcap_lookupnet() succeeded. subNet=172840192, netMask=4294967040
2012-03-22 16:04:55:968 DEBUG [0x884] SnifferSession.cpp[252] SnifferSession::PacketFilter_setFilt
 filter set to (udp and ((ip host 10.77.85.103 and port 24606) or (ip host 10.77.85.104 and port 2
2012-03-22 16:04:55:970 TRACE [0x884] CFCDMSnifferSession.cpp[245] CFCDMSnifferSession::_snifferSe
 Capturing and forwarding packets.
```

With CAD tracing level set to TRACE, the Agent.dbg log will show every RTP packet being captured. The log will fill quickly, roughly 10MB in 45 seconds.

Agent.dbg

```
2012-03-22 16:04:56:125 TRACE [0x884] filterPacket.cpp[67] FilterPacket::parsePacket: Begin.
2012-03-22 16:04:56:129 DUMP [0x884] filterPacket.cpp[96] FilterPacket::parsePacket: hexPacket: 6E
2012-03-22 16:04:56:130 TRACE [0x884] filterPacket.cpp[106] FilterPacket::parsePacket: destMacAddr
2012-03-22 16:04:56:130 TRACE [0x884] filterPacket.cpp[115] FilterPacket::parsePacket: sourceMacAd
2012-03-22 16:04:56:131 TRACE [0x884] filterPacket.cpp[143] FilterPacket::parsePacket: ipHeaderLen
2012-03-22 16:04:56:131 TRACE [0x884] filterPacket.cpp[163] FilterPacket::parsePacket: sourceIpAdd
2012-03-22 16:04:56:131 TRACE [0x884] filterPacket.cpp[164] FilterPacket::parsePacket: sourceIpAdd
2012-03-22 16:04:56:133 TRACE [0x884] filterPacket.cpp[179] FilterPacket::parsePacket: destination
2012-03-22 16:04:56:134 TRACE [0x884] filterPacket.cpp[180] FilterPacket::parsePacket: destination
2012-03-22 16:04:56:134 TRACE [0x884] filterPacket.cpp[188] FilterPacket::parsePacket: source port
2012-03-22 16:04:56:134 TRACE [0x884] filterPacket.cpp[192] FilterPacket::parsePacket: destination
2012-03-22 16:04:56:135 TRACE [0x884] filterPacket.cpp[196] FilterPacket::parsePacket: UDP packet
2012-03-22 16:04:56:135 TRACE [0x884] filterPacket.cpp[203] FilterPacket::parsePacket: Bytes to se
2012-03-22 16:04:56:135 TRACE [0x884] filterPacket.cpp[216] FilterPacket::parsePacket: Payload typ
2012-03-22 16:04:56:136 TRACE [0x884] filterPacket.cpp[67] FilterPacket::parsePacket: End.
```

If CAD is successful in sending the captured packet to the Recording Service, the following entry will be present after the packet print out above. The syntax **forwardPacketToRecipients: send (inbound)** and **forwardPacketToRecipients: Incoming: Sent 172 bytes** indicates that CAD is sending an INBOUND stream RTP packet. Corresponding OUTBOUND stream RTP packets will have the syntax **forwardPacketToRecipients: send (outbound)** and **forwardPacketToRecipients: Outgoing: Sent 172 bytes**.

Agent.dbg

```
2012-03-22 16:04:56:145 TRACE [0x884] filterPacket.cpp[383] FilterPacket::forwardPacketToRecipient
 send (inbound) [10.77.85.104 24584] to [85104 [ destination [, 10.77.85.100, 3500,3501, false] 3,
2012-03-22 16:04:56:148 TRACE [0x884] filterPacket.cpp[384] FilterPacket::forwardPacketToRecipient
```

When the Recoding Service receives an RTP packet, the following events indicate that it is being written to the .Raw file. Note that there are not any unique indicators to explain which agent, recording session, or call this packet is associated with. Best to isolate reproduction efforts on the Recording Service server when troubleshooting this step.

RPServer.dbg

```
2012-03-22 16:04:35:017 CALL [0x281aba0] VoiceFile.cpp[677] WriteRTP: Begin.
2012-03-22 16:04:35:017 TRACE [0x281aba0] VoiceFile.cpp[728] FilterOut: The packet got through.
2012-03-22 16:04:35:017 CALL [0x281aba0] VoiceFile.cpp[923] WriteVoice: Begin.
2012-03-22 16:04:35:017 TRACE [0x281aba0] VoiceFile.cpp[925] WriteVoice: Size of the RTP packet is
2012-03-22 16:04:35:017 TRACE [0x281aba0] VoiceFile.cpp[751] NeedNewHistogramEntry: Sequence = 137
2012-03-22 16:04:35:017 TRACE [0x281aba0] VoiceFile.cpp[755] NeedNewHistogramEntry:
 Timestamp: Current = 1879022600, Previous = 1879022440, Diff = 20
2012-03-22 16:04:35:017 CALL [0x281aba0] VoiceFile.cpp[184] IsEnoughRoom: Begin.
2012-03-22 16:04:35:017 CALL [0x281aba0] VoiceFile.cpp[184] IsEnoughRoom: End.
2012-03-22 16:04:35:017 TRACE [0x281aba0] VoiceFile.cpp[973] WriteVoice: Buffering.
2012-03-22 16:04:35:017 CALL [0x281aba0] VoiceFile.cpp[191] Pack: Begin.
2012-03-22 16:04:35:017 CALL [0x281aba0] VoiceFile.cpp[191] Pack: End.
2012-03-22 16:04:35:017 CALL [0x281aba0] VoiceFile.cpp[191] Pack: Begin.
2012-03-22 16:04:35:017 CALL [0x281aba0] VoiceFile.cpp[191] Pack: End.
2012-03-22 16:04:35:017 CALL [0x281aba0] VoiceFile.cpp[191] Pack: Begin.
2012-03-22 16:04:35:017 CALL [0x281aba0] VoiceFile.cpp[191] Pack: End.
2012-03-22 16:04:35:017 CALL [0x281aba0] VoiceFile.cpp[923] WriteVoice: End.
2012-03-22 16:04:35:017 CALL [0x281aba0] VoiceFile.cpp[677] WriteRTP: End.
```

Once the recording session has ended, CAD will send the RASCAL Service metadata pertaining to the session. Here we see the reception and subsequent query for the metadata.

FCRasSvr.dbg

```
2012-03-22 16:04:56:907 CALL [0x3098ba0] CFcvrsCritSection.cpp[315] writeRecordingData: Begin. Glo
 userName(agent2), userType(1), userTeam(Default), userExtension(85104), initiatorID(1000), initia
2012-03-22 16:04:56:907 CALL [0x3098ba0] CFcvrsCritSectionOther.cpp[30] getGlobalID: ENTRY
2012-03-22 16:04:56:907 DEBUG [0x3098ba0] CFcvrsCritSectionOther.cpp[35] getGlobalID: Database not
2012-03-22 16:04:56:907 CALL [0x3098ba0] CFcvrsCritSectionOther.cpp[60] getGlobalID: END globalID=
2012-03-22 16:04:56:907 CALL [0x3098ba0] RecordLogFinder.cpp[141] insertRecordLog:
 ENTRY GlobalID={'14ff20524153000009d80011', 1332439149}, recordStartTime=1332446672, userName=(ag
 initiatorID=(1000), initiatorType=2, recordID=3, audioFile=(10.77.85.100:201203221604540000218510
2012-03-22 16:04:56:911 DEBUG [0x2d7fba0] ODBCConnection.cpp[370] executeQueryStatement:
 query duration [0] text [INSERT INTO FCRasRecordLog (dcServerID,dnGlobalID,dnRecordStartTime,dnRe
 dcRecordUserName,dcRecordUserTeam,dcRecordInitiatorID,dnRecordInitiatorType,dcAudioFileName,dfAud
 VALUES ('14ff20524153000009d80011', 1332439149, 1332446672, 1332446696, 4, 3, 'agent2','Default',
  '10.77.85.100:201203221604540000218510410000',0, 0)]  returns 0 ([0,1,100] = succeeded, other = f
```