Debug Command Output on Cisco IOS Voice Gateways

The debugging capability for Cisco voice gateways enables you to identify and track a specific call in a multiple-call environment. This capability allows you to correlate call information between gateways or to identify specific debug messages associated with a single call when multiple voice calls were simultaneously active.

Voice debug output contains a standardized header to the debug outputs of multiple voice modules, such as voice telephony service provider (VTSP), call control application program interface (CCAPI), session application (SSAPP), and interactive voice response (IVR).

The following information can be found in the *Cisco IOS Debug Command Reference*:

- Using debug commands is contained in the "Using Debug Commands" chapter.
- Conditionally triggered debugging is contained in the "Conditionally-Triggered Debugging" chapter.
- Details about individual **debug** commands are listed.

This chapter contains the following information:

Voice Debug Concepts

The following concepts provide background for the enhanced debug capabilities on Cisco voice gateways:

Americas Headquarters:
Cisco Systems, Inc., 170 West Tasman Drive, San Jose, CA 95134-1706 USA

Debug Header Format

You can control the contents of the standardized header with the following display options:

- Short 6-byte global unique identifier (GUID)
- Full 16-byte GUID
- Short header that contains only the CallEntry ID

The format of the GUID headers is as follows:
//CallEntryID/GUID/Module-dependent-list/Function-name:

The format of the short header is as follows:
//CallEntryID/Function-name:

The parameters in the header are as follows:

- CallEntryID—Numerically identifies a specific call leg such as an incoming ISDN call or an outgoing H.323 call. The CallentryID ranges from 1 to 32767. When the CallEntryID is not available to the function producing the debug message, the header displays the CallEntryID field as "-1."
- GUID—Each call is assigned a GUID, which is generally used for billing purposes. The GUID is a 16-byte quantity that uniquely identifies a call throughout the entire network and over time. Gateway information and time stamp are embedded in the GUID. The GUID is sometimes called a conference point. See the "CallEntry ID and GUID Call Legs" section on page 4 for an example of the GUID and CallEntry ID call legs.

  By default, the debug header displays a more compact 6-byte form of the GUID that is generally unique enough for debugging purposes. If you need to correlate GUID information to third-party devices or to conduct debugging sessions lasting more than a month, you can display the full 16-byte header with the **voice call debug** command using the **full-guid** keyword.

  When the GUID information is unavailable to the module producing the **debug** message, the header displays the GUID field as blank or filled with "x" characters (/xxxxxxx/). For non-IVR debugs, if the GUID field is unavailable, the header is //CallentryID/xxxxxxxx/. For IVR debugs, the header is //CallentryID//. IVR rarely has GUID information but reserves the space in order to have field-by-field consistency.
- Module-dependent-list—These parameters are dependent on which module is used. The dependent parameters for the various modules are shown in Table 1.

Table 1 Module-Dependent List Parameters

Module	Dependent Parameters
CCAPI	None.
VTSP	Port name, channel number, DSP slot, and DSP channel number.
SSAPP	Dial-peer number and the number of the connection through the dial peer. (Multiple sessions can be associated with a single dial peer.)
IVR	See the "IVR Module-Dependent List" section on page 3.

- Function-name—Identifies call progress though the internal modules of the Cisco IOS code. This is free-form debugging used by developers.

IVR Module-Dependent List

The module-dependent list for IVR information has the following form:
<Module Name> : [LP:] [<OBJ><OBJ-ID>: [<OBJ><OBJ-ID>:[...]]]

Module names are shown in Table 2. The LP parameter is the *link point*, which is the point where two or more objects can be associated . The OBJ parameter is the object name, shown in Table 3. The OBJ-ID is a numeric identifier for the object. The ranges are also shown in Table 3.

Table 2 IVR Module Names

Module Name	Module
AAPL	Application infrastructure
DCM	Digit collect
DPM	Dynamic prompt
MCM	Media content
MSM	Media stream
MSW	Media service wrapper
PCM	Place call (call setup)
RTSP	Real-time streaming protocol (RTSP) client
TCL2	Tcl IVR 2

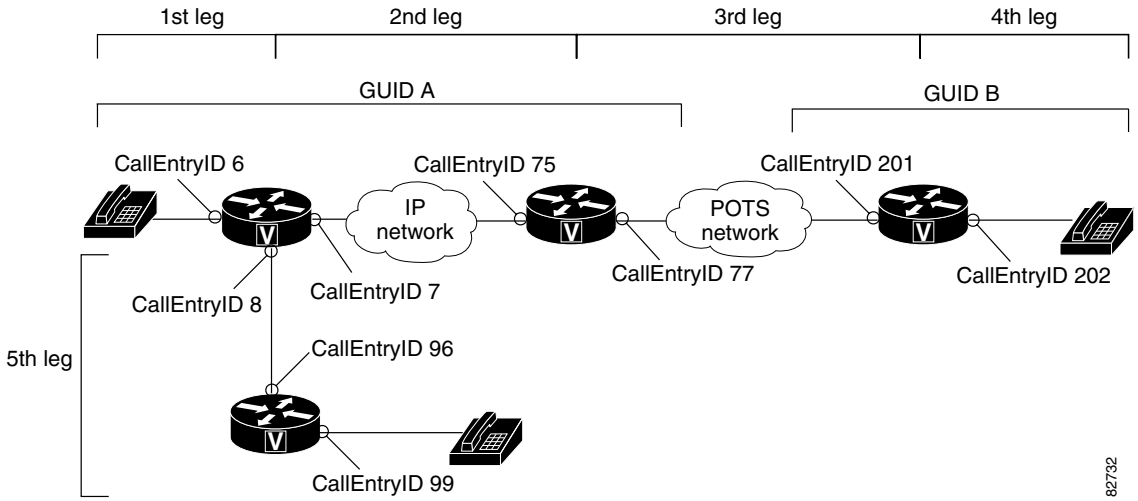
Table 3 Object Names and Identifier Values

Object Name	Description	Identifier Range
CN	Connection	–2147483647 to 2147483647 with 0 and –1 being invalid (either) or unavailable (latter)
DP	Dynamic prompt	1 to –4294967295 or 0 (invalid or unavailable)
GS	Generic stream	1 to –4294967295 or 0 (invalid or unavailable)
HN	Handler	0 to –4294967295
LG	Call leg	1 to –2147483647 or –1 (invalid or unavailable)
MC	Media content	1 to –4294967295 or 0 (invalid or unavailable)
MR	Media content reader	1 to –4294967295 or 0 (invalid or unavailable)
MS	Media stream	1 to –4294967295 or 0 (invalid or unavailable)
RS	RTSP session	1 to –4294967295 or 0 (invalid or unavailable)

CallEntry ID and GUID Call Legs

The parts of the call that are identified by CallEntry IDs and GUIDs are shown in Figure 8. The example shown in the figure is a conference call in which two of the participants are on the same IP network and the third participant is called over the POTS network. Most of the IP-based voice signaling protocols (like H.225) carry the GUID information between gateways. Traditional voice connections such as POTS and ISDN have no means to carry a GUID. A voice gateway creates a new GUID when one is not available from the call originator. In the example, call legs 1, 2, and 5 use GUID A, and leg 4 uses GUID B. Leg 3 is across the POTS network, so no GUID is associated with it. There is a CallEntry ID for each incoming or outgoing call on each gateway.

Figure 8 GUIDs and CallEntry IDs in a Conference Call



Enabling Command Profile Debugging

You can enable a set of debugs based upon a specific profile. You can enable the following profiles:

- Fax Debug Profile
- Modem Debug Profile
- Voice Debug Profile

Enabling the debugging on each of these profiles produces many types of debug information. The advantage of using profile debugging is that instead of having to enable several debug commands to get a full picture of the problem, you can enable a profile which gives you all of the debugs that are appropriate.

When profile debugging is enabled, a large number of debug messages are produced which might affect system performance. For this reason, you should only use profile debugging during low traffic periods or on a non-production system.

Caution The **debug voip profile** commands generate debug messages from many VoIP components, which generates a large number of debug messages. The number of messages can cause a performance impact on your router. This command should only be used during low traffic periods.

Fax Debug Profile

The debug commands that are activated within the fax debug profile depends on the syntax that you use when you enable the command. The syntax for the fax debug profile is as follows:

debug voip profile fax { **mail** | **relay** { **application** | **signaling** }}

For more information on command syntax, refer to the *Cisco IOS Debug Command Reference*.

If you use the **debug voip profile fax mail** command, you enable the following debug commands for an onramp or offramp fax mail call:

- **debug crm all**
- **debug csm voice**
- **debug fax fmsp all**
- **debug fax mta all**
- **debug fax mmoip aaa**
- **debug isdn q931**
- **debug tgrm all**
- **debug voip application all**
- **debug voip application vxml all**
- **debug voip ccapi all**
- **debug voip dsm all**
- **debug voip dspapi all**
- **debug voip hpi all**
- **debug voip ivr all**
- **debug voip vtsp all**

The following debug commands are enabled for access servers with MICA modem cards:

- **debug fax receive all**
- **debug fax send all**
- **debug fax fmail client**
- **debug fax fmail server**
- **debug fax text-to-fax**
- **debug fax tiff reader**
- **debug fax tiff writer**

The following debug options are enabled for access servers with universal port dial feature cards:

- **debug fax fmsp all**
- **debug fax foip all**

- **debug fax dmsp all**
- **debug fax mspi all**
- **debug voip application vxml all**
- **debug voip ivr all**

If you use the **debug voip profile fax relay** command, you enable the **debug fax relay t30 all-level-1** and the sets specified by either the **application** or **signaling** keyword.

For the **debug voip profile fax relay application** command, the following debugs are enabled for fax relay applications:

- **debug voip application all**
- **debug voip application vxml all**
- **debug voip ccapi all**
- **debug voip dialpeer all**
- **debug voip ivr all**

For the **debug voip profile fax relay signaling** command, the following debugs are enabled for fax relay signaling:

- **debug cch323 all**
- **debug ccsip error**
- **debug ccsip messages**
- **debug cdapi detail**
- **debug cdapi events**
- **debug crm all**
- **debug csm voice**
- **debug gtd error**
- **debug gtd events**
- **debug h225 asn1**
- **debug h225 events**
- **debug h225 q931**
- **debug h245 asn1**
- **debug h245 event**
- **debug isdn q931**
- **debug mgcp errors**
- **debug mgcp events**
- **debug mgcp media**
- **debug mgcp packets**
- **debug mgcp voipcac**
- **debug rtpspi all**
- **debug tgrm all**
- **debug voip ccapi all**

- **debug voip dsm all**
- **debug voip dspapi all**
- **debug voip hpi all**
- **debug voip rawmsg**
- **debug voip tsp all**
- **debug voip vtsp all**

For detailed information about the individual debug commands, refer to the *Cisco IOS Debug Command Reference*.

Modem Debug Profile

The debug commands that are activated within the modem debug profile depends on the syntax that you use when you enable the command. The syntax for the modem debug profile is as follows:

debug voip profile modem { **pass-through signaling** | **relay signaling** }

For more information on command syntax, refer to the *Cisco IOS Debug Command Reference*.

If you use the **debug voip profile modem pass-through signaling** command, you enable the following debug commands for modem pass-through signaling:

- **debug cch323 all**
- **debug ccsip error**
- **debug ccsip messages**
- **debug cdapi detail**
- **debug cdapi events**
- **debug csm voice**
- **debug crm all**
- **debug gtd error**
- **debug gtd events**
- **debug h225 asn1**
- **debug h225 events**
- **debug h225 q931**
- **debug isdn q931**
- **debug mgcp errors**
- **debug mgcp events**
- **debug mgcp media**
- **debug mgcp packets**
- **debug mgcp voipcac**
- **debug rtpspi all**
- **debug tgrm all**
- **debug voip ccapi all**
- **debug voip dsm all**

- **debug voip dspapi all**
- **debug voip hpi all**
- **debug voip rawmsg**
- **debug voip tsp all**
- **debug voip vtsp all**

If you use the **debug voip profile modem relay signaling** command, you enable the following debug commands for modem relay signaling:

- **debug cch323 all**
- **debug ccsip error**
- **debug ccsip messages**
- **debug cdapi detail**
- **debug cdapi events**
- **debug csm voice**
- **debug crm all**
- **debug h245 asn1**
- **debug h245 event**
- **debug isdn q931**
- **debug mgcp errors**
- **debug mgcp events**
- **debug mgcp media**
- **debug mgcp packets**
- **debug mgcp voipcac**
- **debug tgrm all**
- **debug voip ccapi all**
- **debug voip dsm all**
- **debug voip dspapi all**
- **debug voip hpi all**
- **debug voip tsp all**
- **debug voip vtsp all**

For detailed information about the individual debug commands, refer to the *Cisco IOS Debug Command Reference*.

Voice Debug Profile

The debug commands that are activated within the voice debug profile depends on the syntax that you use when you enable the command. The syntax for the voice debug profile is as follows:

debug voip profile voice { **application** | **signaling** }

For more information on command syntax, refer to the *Cisco IOS Debug Command Reference*.

If you use the **debug voip profile voice application** command, you enable the following debug commands for voice applications:

- **debug voip application all**
- **debug voip application vxml all**
- **debug voip ccapi all**
- **debug voip dialpeer all**
- **debug voip ivr all**

If you use the **debug voip profile voice signaling** command, you enable the following debug commands for voice signaling:

- **debug cch323 all**
- **debug ccsip error**
- **debug ccsip messages**
- **debug cdapi detail**
- **debug cdapi events**
- **debug crm all**
- **debug csm voice**
- **debug gtd error**
- **debug gtd events**
- **debug h225 asn1**
- **debug h225 events**
- **debug h225 q931**
- **debug h245 asn1**
- **debug h245 event**
- **debug isdn q931**
- **debug mgcp errors**
- **debug mgcp events**
- **debug mgcp media**
- **debug mgcp packets**
- **debug mgcp voipcac**
- **debug rtpspi all**
- **debug tgrm all**
- **debug voip ccapi all**
- **debug voip dsm all**
- **debug voip dspapi all**
- **debug voip hpi all**
- **debug voip rawmsg**
- **debug voip tsp all**
- **debug voip vtsp all**

For detailed information about the individual debug commands, refer to the *Cisco IOS Debug Command Reference*.

Enabling Individual Debug Commands for CCAPI, TSP, and VTSP Debugging

If you want to use a debug command for CCAPI, TSP, or VTSP and you want to limit the output so that the performance on the router is not impacted, you can select to use individual parts of the debug command. You can use **debug voip ccapi individual** *range*, **debug voip tsp individual** *range*, or **debug voip vtsp individual** *range* to get individual debug output for each of these processes. Table 4, Table 5, and Table 6 show the specific debugs for the individual ranges.

Table 4 CCAPI Individual Debug Values

Value	CCAPI Debug Function
1	CC_IDMSG_API_DISPLAY_IES
2	CC_IDMSG_SETUP_IND_COMM_2
3	CC_IDMSG_SETUP_IND_COMM_3
4	CC_IDMSG_SETUP_IND_COMM_4
5	CC_IDMSG_ALERT_IND_5
6	CC_IDMSG_ALERT_IND_6
7	CC_IDMSG_CONNECT_IND_7
8	CC_IDMSG_CONNECT_IND_8
9	CC_IDMSG_RECONNECT_IND_9
10	CC_IDMSG_DISCONNECTED_IND_10
11	CC_IDMSG_DISCONNECTED_IND_11
12	CC_IDMSG_DISCONNECTED_IND_12
13	CC_IDMSG_DISCONNECT_DONE_IND_13
14	CC_IDMSG_DISCONNECT_DONE_IND_14
15	CC_IDMSG_DISCONNECT_DONE_IND_15
16	CC_IDMSG_PRE_DISC_CAUSE_16
17	CC_IDMSG_PRE_DISC_CAUSE_17
18	CC_IDMSG_DIGIT_BEGIN_IND_18
19	CC_IDMSG_DIGIT_END_IND_19
20	CC_IDMSG_DIGIT_END_IND_20
21	CC_IDMSG_DIGIT_END_NO_TERM_21
22	CC_IDMSG_TONE_IND_22
23	CC_IDMSG_FEATURE_IND_23
24	CC_IDMSG_MODIFY_DONE_IND_24
25	CC_IDMSG_MODIFY_MODE_DONE_IND_25
26	CC_IDMSG_INBAND_MSG_RCVD_IND_26
27	CC_IDMSG_INBAND_MSG_DONE_IND_27

Table 4 CCAPI Individual Debug Values (continued)

Value	CCAPI Debug Function
28	CC_IDMSG_UPD_CALL_INFO_IND_28
29	CC_IDMSG_GEN_NTK_ALERT_EVENT_29
30	CC_IDMSG_VOICE_MODE_EVENT_30
31	CC_IDMSG_VOICE_MODE_EVENT_31
32	CC_IDMSG_DIALING_COMPLETE_IND_32
33	CC_IDMSG_DIGITS_DONE_IND_33
34	CC_IDMSG_DIGITS_DONE_IND_34
35	CC_IDMSG_VBD_XMIT_DONE_IND_35
36	CC_IDMSG_FWD_SETUP_IND_36
37	CC_IDMSG_RSVP_DONE_IND_37
38	CC_IDMSG_AUDIT_RSP_IND_38
39	CC_IDMSG_XFR_STATUS_IND_39
40	CC_IDMSG_XFR_STATUS_IND_40
41	CC_IDMSG_XFR_DONE_IND_41
42	CC_IDMSG_XFR_DONE_IND_42
43	CC_IDMSG_XFR_DONE_IND_43
44	CC_IDMSG_TGT_CID_ACTIVE_RCD_44
45	CC_IDMSG_MODIFY_MEDIA_IND_45
46	CC_IDMSG_MODIFY_MEDIA_ACK_IND_46
47	CC_IDMSG_MODIFY_MEDIA_REJ_IND_47
48	CC_IDMSG_MODEM_CALL_START_IND_48
49	CC_IDMSG_MODEM_CALL_DONE_IND_49
50	CC_IDMSG_ACCT_STATUS_IND_50
51	CC_IDMSG_NW_STATUS_IND_51
52	CC_IDMSG_DESTINFO_IND_52
53	CC_IDMSG_LOOPBACK_DONE_IND_53
54	CC_IDMSG_RT_PACKET_STATS_IND_54
55	CC_IDMSG_CUT_PROGRESS_IND_55
56	CC_IDMSG_CUT_PROGRESS_IND_56
57	CC_IDMSG_PROCEEDING_IND_57
58	CC_IDMSG_FACILITY_IND_58
59	CC_IDMSG_INFO_IND_59
60	CC_IDMSG_PROGRESS_IND_60
61	CC_IDMSG_USERINFO_IND_61
62	CC_IDMSG_DISC_PROG_IND_62
63	CC_IDMSG_DISC_PROG_IND_63

Table 4 CCAPI Individual Debug Values (continued)

Value	CCAPI Debug Function
64	CC_IDMSG_PING_DONE_IND_64
65	CC_IDMSG_COT_TEST_DONE_IND_65
66	CC_IDMSG_PROCESS_DONE_IND_66
67	CC_IDMSG_ASSOCIATED_IND_67
68	CC_IDMSG_SUSPEND_IND_68
69	CC_IDMSG_SUSPEND_ACK_IND_69
70	CC_IDMSG_SUSPEND_REJ_IND_70
71	CC_IDMSG_RESUME_IND_71
72	CC_IDMSG_RESUME_ACK_IND_72
73	CC_IDMSG_RESUME_REJ_IND_73
74	CC_IDMSG_IF_SETUP_REQ_PRIV_74
75	CC_IDMSG_IF_SETUP_REQ_PRIV_75
76	CC_IDMSG_IF_ALLOCATE_DSP_76
77	CC_IDMSG_CONNECT_77
78	CC_IDMSG_CONNECT_78
79	CC_IDMSG_PING_79
80	CC_IDMSG_DISCONNECT_80
81	CC_IDMSG_DISCONNECT_81
82	CC_IDMSG_DISCONNECT_82
83	CC_IDMSG_ALERT_83
84	CC_IDMSG_ALERT_84
85	CC_IDMSG_CUT_PROGRESS_85
86	CC_IDMSG_CUT_PROGRESS_86
87	CC_IDMSG_CUT_PROGRESS_87
88	CC_IDMSG_DISC_PROG_88
89	CC_IDMSG_DISC_PROG_89
90	CC_IDMSG_SET_PEER_90
91	CC_IDMSG_SET_PEER_91
92	CC_IDMSG_PROCEEDING_92
93	CC_IDMSG_SETUP_REQ_93
94	CC_IDMSG_SETUP_REQ_94
95	CC_IDMSG_SETUP_REQ_95
96	CC_IDMSG_SETUP_REQ_96
97	CC_IDMSG_SETUP_REQ_97
98	CC_IDMSG_SETUP_REQ_98
99	CC_IDMSG_SETUP_REQ_99

Table 4 CCAPI Individual Debug Values (continued)

Value	CCAPI Debug Function
100	CC_IDMSG_SETUP_REQ_100
101	CC_IDMSG_SETUP_REQ_101
102	CC_IDMSG_SETUP_ACK_102
103	CC_IDMSG_FACILITY_103
104	CC_IDMSG_TRANSFER_REQ_104
105	CC_IDMSG_GET_CONSULT_ID_105
106	CC_IDMSG_FORWARD_TO_106
107	CC_IDMSG_INFO_107
108	CC_IDMSG_NOTIFY_108
109	CC_IDMSG_PROGRESS_109
110	CC_IDMSG_PRE_DISC_110
111	CC_IDMSG_PRE_DISC_111
112	CC_IDMSG_USER_INFO_112
113	CC_IDMSG_MODIFY_113
114	CC_IDMSG_DIGIT_114
115	CC_IDMSG_DIGIT_DIAL_115
116	CC_IDMSG_DIGIT_DIAL_STOP_116
117	CC_IDMSG_FEATURE_117
118	CC_IDMSG_FEATURE_ENABLE_118
119	CC_IDMSG_ASSOCIATE_STREAM_119
120	CC_IDMSG_ASSOCIATE_STREAM_120
121	CC_IDMSG_DISASSOCIATE_STREAM_121
122	CC_IDMSG_DISASSOCIATE_STREAM_122
123	CC_IDMSG_GENERATE_TONE_INFO_123
124	CC_IDMSG_SET_DIGIT_TIMEOUTS_124
125	CC_IDMSG_SET_DIGIT_TIMEOUTS_125
126	CC_IDMSG_SUSPEND_126
127	CC_IDMSG_SUSPEND_ACK_127
128	CC_IDMSG_SUSPEND_REJ_128
129	CC_IDMSG_RESUME_129
130	CC_IDMSG_RESUME_ACK_130
131	CC_IDMSG_RESUME_REJ_131
132	CC_IDMSG_UPDATE_REDIRECT_NUM_132
133	CC_IDMSG_BABBLER_AUDIT_133
134	CC_IDMSG_CONFERENCE_CREATE_134
135	CC_IDMSG_CONFERENCE_CREATE_135

Table 4 CCAPI Individual Debug Values (continued)

Value	CCAPI Debug Function
136	CC_IDMSG_CONFERENCE_CREATE_136
137	CC_IDMSG_CONFERENCE_DESTROY_137
138	CC_IDMSG_CONFERENCE_DESTROY_138
139	CC_IDMSG_CONFERENCE_DESTROY_139
140	CC_IDMSG_LOOPBACK_140
141	CC_IDMSG_COT_TEST_141
142	CC_IDMSG_HANDOFF_142
143	CC_IDMSG_APP_RETURN_143
144	CC_IDMSG_T38_FAX_START_144
145	CC_IDMSG_T38_FAX_DONE_145

Table 5 TSP Individual Debug Values

Value	TSP Debug Function
1	INDIVIDUAL_TSP_DEBUG_TDM_HAIRPIN_CONNECT_001
2	INDIVIDUAL_TSP_DEBUG_TDM_HAIRPIN_DISCONNECT_002
3	INDIVIDUAL_TSP_DEBUG_CCRAWMSG_ENCAP_003
4	INDIVIDUAL_TSP_DEBUG_CDAPI_FORM_MSG_BASIC_SS_INFO_004
5	INDIVIDUAL_TSP_DEBUG_CDAPI_FORM_MSG_005
6	INDIVIDUAL_TSP_DEBUG_CDAPI_FORM_MSG_006
7	INDIVIDUAL_TSP_DEBUG_CDAPI_SEND_MSG_007
8	INDIVIDUAL_TSP_DEBUG_CDAPI_SEND_MSG_008
9	INDIVIDUAL_TSP_DEBUG_CDAPI_SEND_INFO_MSG_009
10	INDIVIDUAL_TSP_DEBUG_ALLOC_CDB_010
11	INDIVIDUAL_TSP_DEBUG_DEALLOC_CDB_011
12	INDIVIDUAL_TSP_DEBUG_CONNECT_IND_012
13	INDIVIDUAL_TSP_DEBUG_CONNECT_IND_EXIT_013
14	INDIVIDUAL_TSP_DEBUG_CONNECT_IND_EXIT_014
15	INDIVIDUAL_TSP_DEBUG_CONNECT_IND_EXIT_015
16	INDIVIDUAL_TSP_DEBUG_CONNECT_IND_EXIT_016
17	INDIVIDUAL_TSP_DEBUG_CONNECT_IND_EXIT_017
18	INDIVIDUAL_TSP_DEBUG_CDAPI_SETUP_ACK_018
19	INDIVIDUAL_TSP_DEBUG_CDAPI_PROCEEDING_019
20	INDIVIDUAL_TSP_DEBUG_CDAPI_ALERT_020
21	INDIVIDUAL_TSP_DEBUG_CDAPI_CONNECT_021
22	INDIVIDUAL_TSP_DEBUG_CDAPI_INFO_022
23	INDIVIDUAL_TSP_DEBUG_CDAPI_PROGRESS_023

Table 5 TSP Individual Debug Values (continued)

Value	TSP Debug Function
24	INDIVIDUAL_TSP_DEBUG_CDAPI_FACILITY_024
25	INDIVIDUAL_TSP_DEBUG_CDAPI_FACILITY_025
26	INDIVIDUAL_TSP_DEBUG_CDAPI_PRE_CONN_DISC_REQ_026
27	INDIVIDUAL_TSP_DEBUG_CDAPI_DISC_PROG_IND_027
28	INDIVIDUAL_TSP_DEBUG_CDAPI_DISCONNECT_REQ_028
29	INDIVIDUAL_TSP_DEBUG_CDAPI_DISCONNECT_REQ_029
30	INDIVIDUAL_TSP_DEBUG_CDAPI_TSP_SS_RESP_030
31	INDIVIDUAL_TSP_DEBUG_CDAPI_TSP_INFO_IND_031
32	INDIVIDUAL_TSP_DEBUG_CDAPI_TSP_PROCEEDING_032
33	INDIVIDUAL_TSP_DEBUG_CDAPI_TSP_ALERT_033
34	INDIVIDUAL_TSP_DEBUG_CDAPI_TSP_ALERT_EXIT_034
35	INDIVIDUAL_TSP_DEBUG_CDAPI_TSP_ALERT_EXIT_035
36	INDIVIDUAL_TSP_DEBUG_CDAPI_TSP_PROGRESS_036
37	INDIVIDUAL_TSP_DEBUG_CDAPI_TSP_INFO_037
38	INDIVIDUAL_TSP_DEBUG_CDAPI_TSP_CONNECT_038
39	INDIVIDUAL_TSP_DEBUG_CDAPI_TSP_CONNECT_CONF_039
40	INDIVIDUAL_TSP_DEBUG_CDAPI_TSP_DISC_PROG_IND_040
41	INDIVIDUAL_TSP_DEBUG_CDAPI_TSP_PROG_IND_PROGRESS_041
42	INDIVIDUAL_TSP_DEBUG_CDAPI_TSP_RELEASE_IND_042
43	INDIVIDUAL_TSP_DEBUG_CDAPI_TSP_RELEASE_IND_EXIT_043
44	INDIVIDUAL_TSP_DEBUG_CDAPI_TSP_RELEASE_COMP_044
45	INDIVIDUAL_TSP_DEBUG_CDAPI_TSP_RELEASE_COMP_CLEAR_045
46	INDIVIDUAL_TSP_DEBUG_SETUP_REQ_EXIT_046
47	INDIVIDUAL_TSP_DEBUG_SETUP_REQ_EXIT_047
48	INDIVIDUAL_TSP_DEBUG_SETUP_REQ_EXIT_048
49	INDIVIDUAL_TSP_DEBUG_TSP_SET_TRANSFER_INFO_049
50	INDIVIDUAL_TSP_DEBUG_TSP_CALL_VOICE_CUT_THROUGH_050
51	INDIVIDUAL_TSP_DEBUG_TSP_CALL_VOICE_CUT_THROUGH_051
52	INDIVIDUAL_TSP_DEBUG_TSP_CALL_VOICE_CUT_THROUGH_052
53	INDIVIDUAL_TSP_DEBUG_TSP_CALL_VOICE_CUT_THROUGH_053
54	INDIVIDUAL_TSP_DEBUG_TSP_MAIN_054
55	INDIVIDUAL_TSP_DEBUG_DO_GLOBAL_END_TO_END_DISC_055
56	INDIVIDUAL_TSP_DEBUG_TSP_CDAPI_MSG_DUMP_056
57	INDIVIDUAL_TSP_DEBUG_TSP_COT_TIMER_START_057
58	INDIVIDUAL_TSP_DEBUG_TSP_COT_TIMER_STOP_058
59	INDIVIDUAL_TSP_DEBUG_TSP_COT_RESULT_059

Table 5 TSP Individual Debug Values (continued)

Value	TSP Debug Function
60	INDIVIDUAL_TSP_DEBUG_TSP_COT_DONE_060
61	INDIVIDUAL_TSP_DEBUG_TSP_COT_TIMEOUT_061
62	INDIVIDUAL_TSP_DEBUG_TSP_COT_REQ_062
63	INDIVIDUAL_TSP_DEBUG_CDAPI_TSP_COT_SETUP_ACK_063
64	INDIVIDUAL_TSP_DEBUG_CDAPI_TSP_RCV_COT_MSG_064
65	INDIVIDUAL_TSP_DEBUG_CDAPI_TSP_RCV_COT_MSG_065
66	INDIVIDUAL_TSP_DEBUG_TSP_CDAPI_PUT_CAUSE_IE_066
67	INDIVIDUAL_TSP_DEBUG_CDAPI_TSP_SETUP_ACK_067
68	INDIVIDUAL_TSP_DEBUG_CDAPI_TSP_RCV_MSG_068

I

Table 6 VTSP Individual Debug Values

Value	VTSP Debug Function
1	INDIVIDUAL_VTSP_DEBUG_SETUP_REQ_PEND_DEFER_001
2	INDIVIDUAL_VTSP_DEBUG_SETUP_REQ_WAIT_PEND_SUCCESS_002
3	INDIVIDUAL_VTSP_DEBUG_SETUP_REQ_WAIT_PEND_FAIL_003
4	INDIVIDUAL_VTSP_DEBUG_TDM_HPM_COMPLETE_004
5	INDIVIDUAL_VTSP_DEBUG_TDM_HPM_COMPLETE_EXIT_005
6	INDIVIDUAL_VTSP_DEBUG_TDM_HPM_CHECK_006
7	INDIVIDUAL_VTSP_DEBUG_TDM_HPM_CHECK_EXIT_007
8	INDIVIDUAL_VTSP_DEBUG_GENERATE_DISC_008
9	INDIVIDUAL_VTSP_DEBUG_GENERATE_DISC_EXIT_009
10	INDIVIDUAL_VTSP_DEBUG_SETUP_IND_ACK_010
11	INDIVIDUAL_VTSP_DEBUG_SETUP_IND_ACK_EXIT_011
12	INDIVIDUAL_VTSP_DEBUG_PROCEEDING_012
13	INDIVIDUAL_VTSP_DEBUG_PRE_CON_DISCONNECT_013
14	INDIVIDUAL_VTSP_DEBUG_PRE_CON_DISCONNECT_EXIT_014
15	INDIVIDUAL_VTSP_DEBUG_SET_DIGIT_TIMEOUTS_015
16	INDIVIDUAL_VTSP_DEBUG_CONNECT_016
17	INDIVIDUAL_VTSP_DEBUG_LOOPBACK_017
18	INDIVIDUAL_VTSP_DEBUG_RING_NOAN_TIMER_018
19	INDIVIDUAL_VTSP_DEBUG_ALERT_CONNECT_019
20	INDIVIDUAL_VTSP_DEBUG_PRE_CON_DISC_REL_EXIT_020
21	INDIVIDUAL_VTSP_DEBUG_HOST_DISC_CLEANUP_021
22	INDIVIDUAL_VTSP_DEBUG_HOST_DISC_CLEANUP_EXIT_022
23	INDIVIDUAL_VTSP_DEBUG_DISCONNECT_023
24	INDIVIDUAL_VTSP_DEBUG_DISCONNECT_EXIT_024

Table 6 VTSP Individual Debug Values (continued)

Value	VTSP Debug Function
25	INDIVIDUAL_VTSP_DEBUG_DISCONNECT_EXIT_025
26	INDIVIDUAL_VTSP_DEBUG_CONNECT_DIAL_026
27	INDIVIDUAL_VTSP_DEBUG_SETUP_PEND_DIAL_027
28	INDIVIDUAL_VTSP_DEBUG_PRE_DISC_CAUSE_028
29	INDIVIDUAL_VTSP_DEBUG_SETUP_PEND_CONNECT_029
30	INDIVIDUAL_VTSP_DEBUG_SETUP_REQ_PEND_FAIL_030
31	INDIVIDUAL_VTSP_DEBUG_SETUP_REQ_DISC_031
32	INDIVIDUAL_VTSP_DEBUG_RELEASE_TIMEOUT_032
33	INDIVIDUAL_VTSP_DEBUG_SETUP_PEND_PROCEEDING_EXIT_033
34	INDIVIDUAL_VTSP_DEBUG_SETUP_PEND_PROCEEDING_EXIT_034
35	INDIVIDUAL_VTSP_DEBUG_PEND_RELEASE_IND_035
36	INDIVIDUAL_VTSP_DEBUG_PEND_RELEASE_IND_EXIT_036
37	INDIVIDUAL_VTSP_DEBUG_DISCONNECT_NO_DSP_CHAN_037
38	INDIVIDUAL_VTSP_DEBUG_DISCONNECT_NO_DSP_CHAN_EXIT_038
39	INDIVIDUAL_VTSP_DEBUG_CALL_FEATURE_IND_039
40	INDIVIDUAL_VTSP_DEBUG_SETUP_PEND_PROGRESS_040
41	INDIVIDUAL_VTSP_DEBUG_SETUP_PEND_ALERT_041
42	INDIVIDUAL_VTSP_DEBUG_SETUP_PEND_ALERT_EXIT_042
43	INDIVIDUAL_VTSP_DEBUG_SETUP_PEND_FIRST_PROGRESS_043
44	INDIVIDUAL_VTSP_DEBUG_SETUP_PEND_FIRST_PROGRESS_EXIT_044
45	INDIVIDUAL_VTSP_DEBUG_SETUP_PEND_FIRST_PROGRESS_EXIT_045
46	INDIVIDUAL_VTSP_DEBUG_SETUP_PEND_PROG_PROCEEDING_046
47	INDIVIDUAL_VTSP_DEBUG_PROCEEDING_R2_PEND_DIAL_047
48	INDIVIDUAL_VTSP_DEBUG_ALERT_R2_PEND_DIAL_048
49	INDIVIDUAL_VTSP_DEBUG_CONN_R2_PEND_DIAL_049
50	INDIVIDUAL_VTSP_DEBUG_SETUP_R2_PEND_DIAL_050
51	INDIVIDUAL_VTSP_DEBUG_R2_PEND_DIAL_ALL_051
52	INDIVIDUAL_VTSP_DEBUG_INFO_IND_052
53	INDIVIDUAL_VTSP_DEBUG_ALERT_053
54	INDIVIDUAL_VTSP_DEBUG_ALERT_EXIT_054
55	INDIVIDUAL_VTSP_DEBUG_PROGRESS_055
56	INDIVIDUAL_VTSP_DEBUG_DISC_PROG_IND_056
57	INDIVIDUAL_VTSP_DEBUG_SETUP_PEND_DISC_PI_IND_057
58	INDIVIDUAL_VTSP_DEBUG_INFO_058
59	INDIVIDUAL_VTSP_DEBUG_FEATURE_059
60	INDIVIDUAL_VTSP_DEBUG_SETUP_PEND_ALERT_NO_TIMEOUT_060

Table 6 VTSP Individual Debug Values (continued)

Value	VTSP Debug Function
61	INDIVIDUAL_VTSP_DEBUG_SETUP_PEND_ALERT_NO_TIMEOUT_EXIT_061
62	INDIVIDUAL_VTSP_DEBUG_CALL_FEATURE_ENABLE_062
63	INDIVIDUAL_VTSP_DEBUG_XCCSM_COT_TEST_DONE_063
64	INDIVIDUAL_VTSP_DEBUG_XCCSM_COT_TEST_TIMEOUT_064
65	INDIVIDUAL_VTSP_DEBUG_XCCSM_COT_TEST_065
66	INDIVIDUAL_VTSP_DEBUG_CALL_FEATURE_066
67	INDIVIDUAL_VTSP_DEBUG_TCSM_COT_TEST_DONE_067
68	INDIVIDUAL_VTSP_DEBUG_TCSM_COT_TEST_TIMEOUT_068
69	INDIVIDUAL_VTSP_DEBUG_TCSM_ACT_COT_TEST_069
70	INDIVIDUAL_VTSP_DEBUG_PLAY_BUSY_TIMER_START_070
71	INDIVIDUAL_VTSP_DEBUG_PLAY_BUSY_TIMER_STOP_071
72	INDIVIDUAL_VTSP_DEBUG_RING_NOAN_TIMER_START_072
73	INDIVIDUAL_VTSP_DEBUG_RING_NOAN_TIMER_STOP_073
74	INDIVIDUAL_VTSP_DEBUG_VTSP_TIMER_074
75	INDIVIDUAL_VTSP_DEBUG_VTSP_TIMER_STOP_075
76	INDIVIDUAL_VTSP_DEBUG_VTSP_ALLOCATE_CDB_076
77	INDIVIDUAL_VTSP_DEBUG_VTSP_DO_CALL_SETUP_IND_077
78	INDIVIDUAL_VTSP_DEBUG_VTSP_DO_CALL_SETUP_IND_EXIT_078
79	INDIVIDUAL_VTSP_DEBUG_VTSP_REQUEST_CALL_079
80	INDIVIDUAL_VTSP_DEBUG_VTSP_REQUEST_CALL_EXIT_080
81	INDIVIDUAL_VTSP_DEBUG_VTSP_REALLOC_CDB_081
82	INDIVIDUAL_VTSP_DEBUG_VTSP_OG_CALL_REQ_EXIT_082
83	INDIVIDUAL_VTSP_DEBUG_VTSP_FREE_CDB_083
84	INDIVIDUAL_VTSP_DEBUG_TGRM_DISC_REL_084
85	INDIVIDUAL_VTSP_DEBUG_VTSP_CC_CALL_DISCONNECTED_085
86	INDIVIDUAL_VTSP_DEBUG_SIGO_BDROP_086
87	INDIVIDUAL_VTSP_DEBUG_SIGO_PRE_CON_DISCONNECT_087
88	INDIVIDUAL_VTSP_DEBUG_SIGO_PROCEEDING_088
89	INDIVIDUAL_VTSP_DEBUG_SIGO_GENERATE_DISC_089
90	INDIVIDUAL_VTSP_DEBUG_SIGO_ALERT_090
91	INDIVIDUAL_VTSP_DEBUG_SIGO_ALERT_CONNECT_091
92	INDIVIDUAL_VTSP_DEBUG_SIGO_SETUP_PEND_CONNECT_092
93	INDIVIDUAL_VTSP_DEBUG_DO_SIGO_CALL_SETUP_REQ_093
94	INDIVIDUAL_VTSP_DEBUG_DO_SIGO_CALL_SETUP_REQ_SESSION_094
95	INDIVIDUAL_VTSP_DEBUG_DSM_MEDIA_EVENT_CB_095
96	INDIVIDUAL_VTSP_DEBUG_DSM_PEER_EVENT_CB_096

Table 6 VTSP Individual Debug Values (continued)

Value	VTSP Debug Function
97	INDIVIDUAL_VTSP_DEBUG_DSM_FEATURE_NOTIFY_CB_097
98	INDIVIDUAL_VTSP_DEBUG_DSM_BRIDGE_CHECK_CB_098
99	INDIVIDUAL_VTSP_DEBUG_DSM_BRIDGE_STATUS_EXIT_099
100	INDIVIDUAL_VTSP_DEBUG_DSM_SET_FAX_FEAT_EXIT_100
101	INDIVIDUAL_VTSP_DEBUG_DS_DO_DIAL_101
102	INDIVIDUAL_VTSP_DEBUG_DS_DIALING_DEFAULT_102

Managing the Voice Call Debug Output

The debug output for calls on Cisco voice gateways is managed by using the **voice call debug** command in global configuration mode. When this command is enabled, the standardized header appears when voice debugs are used.

Supported Commands

The voice debug commands that support the standardized header include the following:

- **debug crm**
- **debug fax dmsp**
- **debug fax fmsp**
- **debug fax foip**
- **debug fax mmoip aaa**
- **debug fax mspi**
- **debug fax mta**
- **debug mgcp all**
- **debug mgcp endpoint**
- **debug mgcp endptdb**
- **debug mgcp errors**
- **debug mgcp events**
- **debug mgcp gcfm**
- **debug mgcp inout**
- **debug mgcp media**
- debug mgcp nas
- debug mgcp packets
- debug mgcp parser
- **debug mgcp src**
- **debug mgcp state**

- **debug mgcp voipcac**
- **debug rtsp all**
- **debug rtsp api**
- **debug rtsp client**
- **debug rtsp error**
- **debug rtsp pmh**
- **debug rtsp session**
- **debug rtsp socket**
- **debug tgrm**
- **debug voip application vxml**
- **debug voip avlist**
- **debug voip ccapi**
- **debug voip dialpeer**
- **debug voip dsm**
- **debug voip dspapi**
- **debug voip hpi**
- **debug voip ivr all**
- **debug voip ivr applib**
- **debug voip ivr callsetup**
- **debug voip ivr digitcollect**
- **debug voip ivr dynamic**
- **debug voip ivr error**
- **debug voip ivr script**
- **debug voip ivr settlement**
- **debug voip ivr states**
- **debug voip ivr tclcommands**
- **debug voip profile fax**
- **debug voip profile help**
- debug voip profile modem
- debug voip profile voice
- debug voip rawmsg
- debug voip tsp
- debug voip vtsp
- debug vtsp all
- debug vtsp dsp
- debug vtsp error
- debug vtsp event
- debug vtsp port

- debug vtsp rtp
- debug vtsp send-nse
- debug vtsp session
- debug vtsp stats
- debug vtsp vofr subframe
- debug vtsp tone

For detailed examples of these debug commands, see the *Cisco IOS Debug Command Reference*.

SUMMARY STEPS

1. **enable**
2. **configure terminal**
3. **voice call debug** {full-guid | short-header}
4. **exit**
5. Run desired debug command.

DETAILED STEPS

	Command or Action	Purpose
Step 1	**enable** **Example:** `Router> enable`	Enables privileged EXEC mode. • Enter your password if prompted.
Step 2	**configure terminal** **Example:** `Router# configure terminal`	Enters global configuration mode.
Step 3	**voice call debug {full-guid | short-header}** **Example:** `Router(config)# voice call debug full-guid`	Specifies the full GUID or short header for debugging a voice call in a multiple-call environment. • **full-guid**—Displays the GUID in a 16-byte header. When the **no** version of this command is input with the **full-guid** keyword, the short 6-byte version displays. This is the default. • **short-header**—Displays the CallEntry ID in the header without displaying the GUID or module-specific parameters.

	Command or Action	Purpose
Step 4	**exit** **Example:** `Router(config)# exit`	Exits to privileged EXEC mode.
Step 5	Run desired voice debug command. **Example:** `Router# debug voip ccapi inout`	Enter the appropriate debug command for your troubleshooting needs. • The voice debug commands that support the standardized header and detailed examples of the individual debug commands are in the *Cisco IOS Debug Command Reference*, Release 12.3.

Sample Output Examples for the Enhanced debug Commands

The following sections provide examples for the enhanced **debug** commands:

- Full GUID Header: Example, page 22
- Short Header: Example, page 23
- Setup and Teardown: Example, page 23

Full GUID Header: Example

The following is sample output from the **voice call debug** command when the **full-guid** keyword is specified:

```
Router(config)# voice call debug full-guid
!
00:05:12: //1/0E2C8A90-BC00-11D5-8002-DACCFDCEF87D/VTSP:(0:D):0:0:4385/vtsp_insert_cdb:
00:05:12: //-1/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx/CCAPI/cc_incr_if_call_volume:
```

In the first line, the following parameters are described:

- 1 denotes the CallEntry ID.
- 0E2C8A90-BC00-11D5-8002-DACCFDCEF87D is the GUID.
- VTSP:(0:D):0:0:4385 is the module and its dependent parameters. In this case, the module is VTSP, the port number is 0:D, the channel number is 0, the DSP slot is 0, and the DSP channel number is 4385.
- vtsp_insert_cdb: is the function name.

In the second line, the following parameters are described:

- -1 indicates that the CallEntry ID for the CCAPI module is unavailable.
- xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx indicates that the GUID is unavailable.
- CCAPI is the module.
- cc_incr_if_call_volume: is the function name.

Short Header: Example

The following is sample output from the **voice call debug** command when the **short-header** keyword is specified:

```
Router(config)# voice call debug short-header
!
00:10:12: //-1/cc_incr_if_call_volume:
00:10:12: //9/vtsp_open_voice_and_set_params:
```

In the first line, the CallEntry ID is not available, and in the following line the CallEntry ID is 9. The remainder of the line displays the function name.

Setup and Teardown: Example

In the following example, the **debug voip ccapi inout** command used with the **voice call debug full-guid** command traces the execution path through the call control API, which serves as the interface between the call session application and the underlying network-specific software. This output shows how calls are being handled by the voice gateway. Using this debug level, you can see the call setup and teardown operations performed on both the telephony and network call legs.

```
Router# debug voip ccapi inout

*Mar  1 15:35:53.588: //-1/xxxxxxxxxxxx/CCAPI/ccTDConstructTDUsrContainer:
usrContainer[0x638C1BF0], magic[FACE0FFF]
*Mar  1 15:35:53.592: //-1/xxxxxxxxxxxx/CCAPI/ccTDUtilAddDataToUsrContainer:
container=0x638C1BF0, tagID=6, dataSize=16, instID=-1,modifier=1
*Mar  1 15:35:53.592: //-1/xxxxxxxxxxxx/CCAPI/ccTDConstructInstanceTDObject:
tdObject[0x638BC1AC], nxtElem[0x0], magic[0xFACE0FFF] tagID[6], dataLen[16], modif[1]
*Mar  1 15:35:53.592: //-1/xxxxxxxxxxxx/CCAPI/ccTDPvtAddObjectToContainer: Adding
tdObject[0x638BC1AC] instID[-1] into container[0x638C1BF0]
*Mar  1 15:35:53.592: //-1/xxxxxxxxxxxx/CCAPI/ccTDUtilAddDataToUsrContainer:
container=0x638C1BF0, tagID=5, dataSize=276, instID=-1,modifier=1
*Mar  1 15:35:53.592: //-1/xxxxxxxxxxxx/CCAPI/ccTDConstructInstanceTDObject:
tdObject[0x63401148], nxtElem[0x0], magic[0xFACE0FFF] tagID[5], dataLen[276], modif[1]
*Mar  1 15:35:53.592: //-1/xxxxxxxxxxxx/CCAPI/ccTDPvtAddObjectToContainer: Adding
tdObject[0x63401148] instID[-1] into container[0x638C1BF0]
```

In the following lines, the CCAPI receives the call setup. The called number is 34999, and the calling number is 55555. The calling number matches dial peer 10002.

```
*Mar  1 15:35:53.592: //-1/xxxxxxxxxxxx/CCAPI/cc_api_display_ie_subfields:
*Mar  1 15:35:53.592: cc_api_call_setup_ind:
*Mar  1 15:35:53.592:  cisco-username=
*Mar  1 15:35:53.596: ----- ccCallInfo IE subfields -----
*Mar  1 15:35:53.596:  cisco-ani=55555
*Mar  1 15:35:53.596:  cisco-anitype=0
*Mar  1 15:35:53.596:  cisco-aniplan=0
*Mar  1 15:35:53.596:  cisco-anipi=0
*Mar  1 15:35:53.596:  cisco-anisi=0
*Mar  1 15:35:53.596:  dest=34999
*Mar  1 15:35:53.596:  cisco-desttype=0
*Mar  1 15:35:53.596:  cisco-destplan=0
*Mar  1 15:35:53.596:  cisco-rdn=
*Mar  1 15:35:53.596:  cisco-rdntype=-1
*Mar  1 15:35:53.596:  cisco-rdnplan=-1
*Mar  1 15:35:53.596:  cisco-rdnpi=-1
*Mar  1 15:35:53.596:  cisco-rdnsi=-1
*Mar  1 15:35:53.596:  cisco-redirectreason=-1
```

```
*Mar  1 15:35:53.596: //-1/xxxxxxxxxxxx/CCAPI/cc_api_call_setup_ind: (vdbPtr=0x637EC1E0,
callInfo={called=34999,called_oct3=0x80,calling=55555,calling_oct3=0x80,calling_oct3a=0x0,
calling_xlated=false,subscriber_type_str=RegularLine,fdest=1,peer_tag=10002,
prog_ind=0,callingIE_present 1, src_route_label=, tgt_route_label=
clid_transparent=0},callID=0x637B4278)

*Mar  1 15:35:53.596: //-1/xxxxxxxxxxxx/CCAPI/cc_api_call_setup_ind:
*Mar  1 15:35:53.596: //-1/xxxxxxxxxxxx/CCAPI/cc_api_call_setup_ind: type 13 , prot 0
*Mar  1 15:35:53.596: //-1/xxxxxxxxxxxx/CCAPI/ccCheckClipClir:
*Mar  1 15:35:53.596: ccCheckClipClir: calling number is: "55555", calling oct3a is: 0x0
*Mar  1 15:35:53.596: //-1/xxxxxxxxxxxx/CCAPI/ccCheckClipClir:
*Mar  1 15:35:53.596: Calling Party number is User Provided
*Mar  1 15:35:53.596: //-1/xxxxxxxxxxxx/CCAPI/ccCheckClipClir:
*Mar  1 15:35:53.596: Leaving ccCheckClipClir
  calling number is: "55555"
  calling oct3 is:  0x80
  calling oct3a is: 0x0
```

In the next line, 44 is the CallEntry ID.

```
*Mar  1 15:35:53.600: //44/xxxxxxxxxxxx/CCAPI/cc_insert_call_entry: Increment call volume:
0
*Mar  1 15:35:53.600: //44/xxxxxxxxxxxx/CCAPI/cc_insert_call_entry: current call volume: 1
*Mar  1 15:35:53.600: //44/xxxxxxxxxxxx/CCAPI/cc_insert_call_entry: entry's incoming TRUE.
*Mar  1 15:35:53.600: //44/xxxxxxxxxxxx/CCAPI/cc_insert_call_entry: is_incomingis TRUE
*Mar  1 15:35:53.600: //-1/xxxxxxxxxxxx/CCAPI/ccTDConstructHashProfileTab:
profileTable[0x6380E11C], numBuckets[11], numEntries[0]
*Mar  1 15:35:53.600: //-1/xxxxxxxxxxxx/CCAPI/ccTDPvtProfileTableBuildManager: Invoking
necessary profileTable updaters...
*Mar  1 15:35:53.600: //-1/xxxxxxxxxxxx/CCAPI/ccTDPvtUpdateProfileTabFromContainer:
Updating profileTable[0x6380E11C] with objects in container[0x638C1BF0]
*Mar  1 15:35:53.600: //-1/xxxxxxxxxxxx/CCAPI/ccTDPvtUpdateProfileTabFromContainer:
obtained key[5] for the tag[6]
*Mar  1 15:35:53.600: //-1/xxxxxxxxxxxx/CCAPI/ccTDPvtAddObjectToProfileBucket:
profileTable[0x6380E11C], tdObject[0x638BC1AC]
*Mar  1 15:35:53.600: //-1/xxxxxxxxxxxx/CCAPI/ccTDPvtUpdateProfileTabFromContainer:
obtained key[0] for the tag[5]
*Mar  1 15:35:53.600: //-1/xxxxxxxxxxxx/CCAPI/ccTDPvtAddObjectToProfileBucket:
profileTable[0x6380E11C], tdObject[0x63401148]
*Mar  1 15:35:53.600: //-1/xxxxxxxxxxxx/CCAPI/ccTDPvtProfileTableBuildManager:
*Mar  1 15:35:53.600: ccTDUtilDumpAllElemInProfileTab: profileTable[0x6380E11C],
 numBuckets[11], numEntries[2]
*Mar  1 15:35:53.600: Bucket { 0 } ------>0x63401148[0x0,t-5,l-276,d-0x63401168,
m-1,u-56153,g-FACE0FFF]
*Mar  1 15:35:53.604:
*Mar  1 15:35:53.604: Bucket { 5 }
------>0x638BC1AC[0x0,t-6,l-16,d-0x638BC1CC,m-1,u-56153,g-FACE0FFF]
*Mar  1 15:35:53.604:
*Mar  1 15:35:53.604: //-1/xxxxxxxxxxxx/CCAPI/ccTDDestructTDUsrContainer:
Container[0x638C1BF0]
*Mar  1 15:35:53.604: //-1/xxxxxxxxxxxx/CCAPI/cc_incr_if_call_volume: not the VoIP or
MMoIP
*Mar  1 15:35:53.608: //-1/xxxxxxxxxxxx/CCAPI/cc_process_call_setup_ind: (event=
0x63073AA0)
```

In the next line, 45F2AAE28044 is the GUID. The tag 10002 entry shows that the incoming dial peer matched the CallEntry ID.

```
*Mar  1 15:35:53.608: //44/45F2AAE28044/CCAPI/cc_process_call_setup_ind: >>>>CCAPI handed
cid 44 with tag 10002 to app "DEFAULT"
*Mar  1 15:35:53.608: //44/xxxxxxxxxxxx/SSAPP:-1:-1/sess_appl:
ev(24=CC_EV_CALL_SETUP_IND), cid(44), disp(0)
*Mar  1 15:35:53.608: //44/xxxxxxxxxxxx/SSAPP:-1:-1/sess_appl: ev(SSA_EV_CALL_SETUP_IND),
cid(44), disp(0)
```

```
*Mar  1 15:35:53.608: //44/xxxxxxxxxxxx/SSAPP:-1:-1/ssaCallSetupInd:
```

The next line shows CallEntry ID in hexadecimal form, 0x2C (44 in decimal). The CallEntry ID and GUID numbers have been identified. The incoming dial peer is 10002.

```
*Mar  1 15:35:53.608: //44/xxxxxxxxxxxx/CCAPI/ccCallSetContext: (callID=0x2C,
context=0x634A430C)

*Mar  1 15:35:53.608: //44/45F2AAE28044/SSAPP:10002:-1/ssaCallSetupInd: cid(44),
st(SSA_CS_MAPPING),oldst(0), ev(24)ev->e.evCallSetupInd.nCallInfo.finalDestFlag = 1
*Mar  1 15:35:53.608: //44/45F2AAE28044/SSAPP:10002:-1/ssaCallSetupInd: src route label=,
tgt route label= tg_label_flag 0x0
*Mar  1 15:35:53.608: //44/45F2AAE28044/SSAPP:10002:-1/ssaCallSetupInd: finalDest
cllng(55555), clled(34999) tgt_route_label()tg_label_flag 0x0
*Mar  1 15:35:53.612: //44/45F2AAE28044/SSAPP:10002:-1/ssaCallSetupInd: cid(44),
st(SSA_CS_CALL_SETTING),oldst(0), ev(24)dpMatchPeersMoreArg result= 0
```

For CallEntry ID 44, two dial-peer tags (10001 and 20002) were matched with called number 34999.

```
*Mar  1 15:35:53.612: //44/45F2AAE28044/SSAPP:10002:-1/ssaDebugPeers: ssaSetupPeer cid(44)
peer list: tag(10001) called number (34999) tag(20002) called number
(34999)
*Mar  1 15:35:53.612: //44/45F2AAE28044/SSAPP:10002:-1/ssaSetupPeer: dialpeer tags in
rotary= 10001  20002
```

The next line shows that five digits were matched for this dial peer and no prefix was added. The encapType (2) entry indicates a VoIP call.

```
*Mar  1 15:35:53.612: //44/45F2AAE28044/SSAPP:10002:-1/ssaSetupPeer: cid(44),
destPat(34999), matched(5), prefix(), peer(637B0984), peer->encapType (2)
*Mar  1 15:35:53.612: //-1/xxxxxxxxxxxx/CCAPI/cc_can_gateway: Call legs: In=6, Out=1
```

The next line shows the voice gateway sending out a call-proceeding message to the incoming call leg with a progress indicator of 0x0.

```
*Mar  1 15:35:53.612: //44/xxxxxxxxxxxx/CCAPI/ccCallProceeding: (callID=0x2C,
prog_ind=0x0)
```

The next line shows the voice gateway sending out the call-setup request to the outgoing call leg. The dial peer is 10001 with the incoming CallEntry ID being 0x2C.

```
*Mar  1 15:35:53.612: //44/xxxxxxxxxxxx/CCAPI/ccCallSetupRequest: (Inbound call= 0x2C,
outbound peer =10001, dest=,
        params=0x63085D80 mode=0, *callID=0x63086314, prog_ind = 0callingIE_present 1)

*Mar  1 15:35:53.612: //44/45F2AAE28044/CCAPI/ccCallSetupRequest:
*Mar  1 15:35:53.612: ccCallSetupRequest numbering_type 0x80
*Mar  1 15:35:53.612: //44/45F2AAE28044/CCAPI/ccCallSetupRequest:
*Mar  1 15:35:53.616: ccCallSetupRequest: calling number is:55555

*Mar  1 15:35:53.616: //44/45F2AAE28044/CCAPI/ccCallSetupRequest: calling oct3ais:0x0

*Mar  1 15:35:53.616: //-1/xxxxxxxxxxxx/CCAPI/ccCheckClipClir:
*Mar  1 15:35:53.616: ccCheckClipClir: calling number is: "55555", calling oct3a is: 0x0
*Mar  1 15:35:53.616: //-1/xxxxxxxxxxxx/CCAPI/ccCheckClipClir:
*Mar  1 15:35:53.616: Calling Party number is User Provided
*Mar  1 15:35:53.616: //-1/xxxxxxxxxxxx/CCAPI/ccCheckClipClir:
*Mar  1 15:35:53.616: Leaving ccCheckClipClir
  calling number is: "55555"
  calling oct3 is:  0x80
  calling oct3a is: 0x0
*Mar  1 15:35:53.616: //44/45F2AAE28044/CCAPI/ccCallSetupRequest: after ccCheckClipClir -
calling oct3a is:0x0
```

The next line shows that all digits are passed.

```
*Mar  1 15:35:53.616: //44/45F2AAE28044/CCAPI/ccCallSetupRequest: dest pattern 34999,
called 34999, digit_strip 0
*Mar  1 15:35:53.616: //44/45F2AAE28044/CCAPI/ccCallSetupRequest:
*Mar  1 15:35:53.616: callingNumber=55555, calledNumber=34999, redirectNumber=
display_info= calling_oct3a=0
*Mar  1 15:35:53.616: accountNumber=,
finalDestFlag=1,guid=45f2.aae2.1571.11cc.8044.95f5.fabb.6b0f
*Mar  1 15:35:53.616: peer_tag=10001
*Mar  1 15:35:53.616: //-1/xxxxxxxxxxxx/CCAPI/cc_api_display_ie_subfields:
*Mar  1 15:35:53.616: ccCallSetupRequest:
*Mar  1 15:35:53.616:  cisco-username=
*Mar  1 15:35:53.616: ----- ccCallInfo IE subfields -----
*Mar  1 15:35:53.616:  cisco-ani=55555
*Mar  1 15:35:53.616:  cisco-anitype=0
*Mar  1 15:35:53.616:  cisco-aniplan=0
*Mar  1 15:35:53.616:  cisco-anipi=0
*Mar  1 15:35:53.616:  cisco-anisi=0
*Mar  1 15:35:53.620:  dest=34999
*Mar  1 15:35:53.620:  cisco-desttype=0
*Mar  1 15:35:53.620:  cisco-destplan=0
*Mar  1 15:35:53.620:  cisco-rdn=
*Mar  1 15:35:53.620:  cisco-rdntype=-1
*Mar  1 15:35:53.620:  cisco-rdnplan=-1
*Mar  1 15:35:53.620:  cisco-rdnpi=-1
*Mar  1 15:35:53.620:  cisco-rdnsi=-1
*Mar  1 15:35:53.620:  cisco-redirectreason=-1

*Mar  1 15:35:53.620: //-1/xxxxxxxxxxxx/CCAPI/ccIFCallSetupRequestPrivate:
(vdbPtr=0x62EC61A4, dest=, callParams={called=34999,called_oct3=0x80,
calling=55555,calling_oct3=0x80, calling_oct3a= 0x0, calling_xlated=false,
subscriber_type_str=RegularLine, fdest=1, voice_peer_tag=10001},mode=0x0)
*Mar  1 15:35:53.620: //-1/xxxxxxxxxxxx/CCAPI/ccIFCallSetupRequestPrivate:
*Mar  1 15:35:53.620: ccIFCallSetupRequestPrivate: src route label  tgt route label
tg_label_flag 0x0
*Mar  1 15:35:53.620: //-1/xxxxxxxxxxxx/CCAPI/ccIFCallSetupRequestPrivate:  vdbPtr type =
1
*Mar  1 15:35:53.620: //-1/xxxxxxxxxxxx/CCAPI/ccIFCallSetupRequestPrivate:
*Mar  1 15:35:53.620: //-1/xxxxxxxxxxxx/CCAPI/ccIFCallSetupRequestPrivate:
(vdbPtr=0x62EC61A4, dest=, callParams={called=34999, called_oct3 0x80,  calling=55555,
calling_oct3 0x80, calling_oct3a 0x0, calling_xlated=false,  fdest=1,
voice_peer_tag=10001}, mode=0x0, xltrc=-5)
*Mar  1 15:35:53.620: //-1/xxxxxxxxxxxx/CCAPI/ccIFCallSetupRequestPrivate:
```

In the next line, outgoing CallEntry ID 45 is bound to the same GUID 45F2AAE28044.

```
*Mar  1 15:35:53.620: //45/45F2AAE28044/CCAPI/cc_insert_call_entry: not incoming entry
*Mar  1 15:35:53.620: //45/45F2AAE28044/CCAPI/cc_insert_call_entry: entry's incoming
FALSE.
*Mar  1 15:35:53.620: //45/45F2AAE28044/CCAPI/cc_insert_call_entry: is_incomingis FALSE
*Mar  1 15:35:53.624: //44/xxxxxxxxxxxx/CCAPI/ccSaveDialpeerTag: (callID=0x2C,
dialpeer_tag=10001)
*Mar  1 15:35:53.624: //45/xxxxxxxxxxxx/CCAPI/ccCallSetContext: (callID=0x2D,
context=0x634A537C) 0x2D (decimal 45 is the second call leg ID).
*Mar  1 15:35:53.624: //44/xxxxxxxxxxxx/CCAPI/ccCallReportDigits: (callID=0x2C,enable=0x0)
```

The next line shows that the voice gateway informs the incoming call leg that digits were forwarded.

```
*Mar  1 15:35:53.624: //44/xxxxxxxxxxxx/CCAPI/cc_api_call_report_digits_done:
(vdbPtr=0x637EC1E0, callID=0x2C, disp=0)
*Mar  1 15:35:53.624: //44/xxxxxxxxxxxx/SSAPP:-1:-1/sess_appl:
ev(54=CC_EV_CALL_REPORT_DIGITS_DONE), cid(44), disp(0)
*Mar  1 15:35:53.624: //44/45F2AAE28044/SS
```

```
Router#APP:10002:-1/ssaTraceSct:
cid(44)st(SSA_CS_CALL_SETTING)ev(SSA_EV_CALL_REPORT_DIGITS_DONE)
oldst(SSA_CS_MAPPING)cfid(-1)csize(0)in(1)fDest(1)
*Mar  1 15:35:53.624: //44/45F2AAE28044/SSAPP:10002:-1/ssaTraceSct:
-cid2(45)st2(SSA_CS_CALL_SETTING)oldst2(SSA_CS_MAPPING)
*Mar  1 15:35:53.624: //44/45F2AAE28044/SSAPP:10002:-1/ssaDebugPeers: ssaReportDigitsDone
cid(44) peer list: tag(20002) called number (34999)
*Mar  1 15:35:53.624: //44/45F2AAE28044/SSAPP:10002:-1/ssaReportDigitsDone: callid=44
Reporting disabled.
*Mar  1 15:35:53.628: //-1/xxxxxxxxxxxx/CCAPI/cc_api_supported_data: data_mode=0x10082
*Mar  1 15:35:53.628: //45/xxxxxxxxxxxx/CCAPI/cc_api_get_ic_leg_obtained_numbers:
callID=0x2D
```

The next two lines show the IP address of the terminating gateway and indicate that the terminating gateway is reached through Ethernet port 0/0.

```
*Mar  1 15:35:53.628: //-1/xxxxxxxxxxxx/CCAPI/cc_incr_if_call_volume: remote IP is
172.31.85.111
*Mar  1 15:35:53.632: //-1/xxxxxxxxxxxx/CCAPI/cc_incr_if_call_volume: hwidb is Ethernet0/0
*Mar  1 15:35:53.632: //-1/xxxxxxxxxxxx/CCAPI/cc_incr_if_call_volume: create entry in
list: 1
*Mar  1 15:35:53.636: //45/xxxxxxxxxxxx/CCAPI/ccTDUtilGetInstanceCount: For tagID[1] of
callID[45]
*Mar  1 15:35:53.636: //45/45F2AAE28044/CCAPI/ccTDPvtProfileTableObjectAccessManager: No
profileTable set for callID[45]
*Mar  1 15:35:53.636: //45/xxxxxxxxxxxx/CCAPI/ccTDUtilGetInstanceCount: For tagID[2] of
callID[45]
*Mar  1 15:35:53.636: //45/45F2AAE28044/CCAPI/ccTDPvtProfileTableObjectAccessManager: No
profileTable set for callID[45]
```

The next line shows that the voice gateway received a call proceeding message from the terminating gateway. The following line shows that the voice gateway received a call alert from the terminating gateway.

```
*Mar  1 15:35:53.740: //45/xxxxxxxxxxxx/CCAPI/cc_api_call_proceeding: (vdbPtr=0x62EC61A4,
callID=0x2D,
      prog_ind=0x0)
*Mar  1 15:35:53.740: //45/xxxxxxxxxxxx/CCAPI/cc_api_call_alert: (vdbPtr=0x62EC61A4,
callID=0x2D, prog_ind=0x0, sig_ind=0x1)
*Mar  1 15:35:53.744: //45/xxxxxxxxxxxx/SSAPP:-1:-1/sess_appl:
ev(21=CC_EV_CALL_PROCEEDING), cid(45), disp(0)
*Mar  1 15:35:53.744: //45/45F2AAE28044/SSAPP:0:-1/ssaTraceSct:
cid(45)st(SSA_CS_CALL_SETTING)ev(SSA_EV_CALL_PROCEEDING)
oldst(SSA_CS_MAPPING)cfid(-1)csize(0)in(0)fDest(0)
*Mar  1 15:35:53.744: //45/45F2AAE28044/SSAPP:0:-1/ssaTraceSct:
-cid2(44)st2(SSA_CS_CALL_SETTING)oldst2(SSA_CS_CALL_SETTING)
*Mar  1 15:35:53.744: //45/45F2AAE28044/SSAPP:0:-1/ssaCallProc:
*Mar  1 15:35:53.744: //44/xxxxxxxxxxxx/CCAPI/ccGetDialpeerTag: (callID=0x2C)
*Mar  1 15:35:53.744: //45/45F2AAE28044/SSAPP:0:-1/ssaIgnore: cid(45),
st(SSA_CS_CALL_SETTING),oldst(1), ev(21)
*Mar  1 15:35:53.744: //45/xxxxxxxxxxxx/SSAPP:-1:-1/sess_appl: ev(7=CC_EV_CALL_ALERT),
cid(45), disp(0)
*Mar  1 15:35:53.744: //45/45F2AAE28044/SSAPP:0:-1/ssaTraceSct:
cid(45)st(SSA_CS_CALL_SETTING)ev(SSA_EV_CALL_ALERT)oldst(SSA_CS_CALL_SETTING)cfid(-1)csize
(0)in(0)fDest(0)
*Mar  1 15:35:53.744: //45/45F2AAE28044/SSAPP:0:-1/ssaTraceSct:
-cid2(44)st2(SSA_CS_CALL_SETTING)oldst2(SSA_CS_CALL_SETTING)
*Mar  1 15:35:53.744: //44/45F2AAE28044/SSAPP:10002:-1/ssaAlert:
*Mar  1 15:35:53.744: //44/xxxxxxxxxxxx/CCAPI/ccGetDialpeerTag: (callID=0x2C)
Router#
```

In the following line, the voice gateway forwarded a call alert to the originating gateway.

```
*Mar  1 15:35:53.744: //44/xxxxxxxxxxxx/CCAPI/ccCallAlert: (callID=0x2C, prog_ind=0x0,
sig_ind=0x1)
Router#
```

The phone is answered at the called number.

```
Router#!call answered
```

The voice gateway receives a connect message from the terminating gateway.

```
*Mar  1 15:36:05.016: //45/xxxxxxxxxxxx/CCAPI/cc_api_call_connected: (vdbPtr=0x62EC61A4,
callID=0x2D), prog_ind = 0
*Mar  1 15:36:05.016: //45/45F2AAE28044/CCAPI/cc_api_call_connected: setting
callEntry->connected to TRUE
```

The next line shows that the call accounting starts. The leg_type=False message means this is for an outgoing call. The line that follows shows that AAA accounting is not configured.

```
*Mar  1 15:36:05.016: //45/45F2AAE28044/CCAPI/cc_api_call_connected: calling accounting
start for callID=45 leg_type=0
*Mar  1 15:36:05.020: //45/xxxxxxxxxxxx/CCAPI/ccCallSetAAA_Accounting: callID=0x2D,
accounting=0
*Mar  1 15:36:05.020: //45/xxxxxxxxxxxx/SSAPP:-1:-1/sess_appl: ev(8=CC_EV_CALL_CONNECTED),
cid(45), disp(0)
*Mar  1 15:36:05.020: //45/45F2AAE28044/SSAPP:0:-1/ssaTraceSct:
cid(45)st(SSA_CS_ALERT_RCVD)ev(SSA_EV_CALL_CONNECTED)
oldst(SSA_CS_CALL_SETTING)cfid(-1)csize(0)in(0)fDest(0)
*Mar  1 15:36:05.020: //45/45F2AAE28044/SSAPP:0:-1/ssaTraceSct:
-cid2(44)st2(SSA_CS_ALERT_RCVD)oldst2(SSA_CS_CALL_SETTING)
*Mar  1 15:36:05.020: //45/45F2AAE28044/SSAPP:0:-1/ssaConnect:
*Mar  1 15:36:05.020: //44/xxxxxxxxxxxx/CCAPI/ccGetDialpeerTag: (callID=0x2C)
```

The next lines show a conference being set up between the two call legs 0x2C and 0x2D. Bridge complete messages are sent to both the terminating and originating gateways.

```
*Mar  1 15:36:05.020: //44/xxxxxxxxxxxx/CCAPI/ccConferenceCreate: (confID=0x63086424,
callID1=0x2C, callID2=0x2D, tag=0x0)
*Mar  1 15:36:05.020: //45/xxxxxxxxxxxx/CCAPI/cc_api_bridge_done:
(confID=0x15,srcIF=0x62EC61A4, srcCallID=0x2D, dstCallID=0x2C, disposition=0, tag=0x0)
*Mar  1 15:36:05.024: //44/xxxxxxxxxxxx/CCAPI/cc_api_bridge_done:
(confID=0x15,srcIF=0x637EC1E0, srcCallID=0x2C, dstCallID=0x2D, disposition=0, tag=0x0)
```

Here, the voice gateway sets up negotiating capability with the originating telephony leg.

```
*Mar  1 15:36:05.024: //44/xxxxxxxxxxxx/CCAPI/cc_api_caps_ind: (dstVdbPtr=0x62EC61A4,
dstCallId=0x2D, srcCallId=0x2C,
     caps={codec=0x2887F, fax_rate=0xBF, vad=0x3, modem=0x2
           codec_bytes=0, signal_type=3})
*Mar  1 15:36:05.024: //44/xxxxxxxxxxxx/CCAPI/cc_api_caps_ind: (Playout: mode 0, initial
60,min 40, max 300)
*Mar  1 15:36:05.024: //44/xxxxxxxxxxxx/SSAPP:-1:-1/sess_appl:
ev(29=CC_EV_CONF_CREATE_DONE), cid(44), disp(0)
*Mar  1 15:36:05.024: //44/45F2AAE28044/SSAPP:10002:21/ssaTraceSct:
cid(44)st(SSA_CS_CONFERENCING)ev(SSA_EV_CONF_CREATE_DONE)
oldst(SSA_CS_CALL_SETTING)cfid(21)csize(2)in(1)fDest(1)
*Mar  1 15:36:05.024: //44/45F2AAE28044/SSAPP:10002:21/ssaTraceSct:
-cid2(45)st2(SSA_CS_CONFERENCING)oldst2(SSA_CS_ALERT_RCVD)
*Mar  1 15:36:05.024: //44/45F2AAE28044/SSAPP:10002:21/ssaConfCreateDone:
*Mar  1 15:36:05.024: //44/xxxxxxxxxxxx/CCAPI/ccCallConnect: (callID=0x2C), prog_ind = 0
*Mar  1 15:36:05.024: //44/45F2AAE28044/CCAPI/ccCallConnect: setting callEntry->connected
to TRUE
```

```
*Mar  1 15:36:05.024: //44/45F2AAE28044/SSAPP:10002:21/ssaDebugPeers: ssaFlushPeerTagQueue
cid(44) peer list: tag(20002) called number (34999)
*Mar  1 15:36:05.028: //-1/xxxxxxxxxxxx/CCAPI/cc_process_notify_bridge_done:
(event=0x63067FC0)
```

The voice gateway sets up negotiating capability with the terminating VoIP leg.

```
*Mar  1 15:36:05.028: //45/xxxxxxxxxxxx/CCAPI/cc_api_caps_ind: (dstVdbPtr=0x637EC1E0,
dstCallId=0x2C, srcCallId=0x2D,
     caps={codec=0x4, fax_rate=0x2, vad=0x2, modem=0x0
           codec_bytes=20, signal_type=2})
*Mar  1 15:36:05.028: //45/xxxxxxxxxxxx/CCAPI/cc_api_caps_ind: (Playout: mode 0, initial
60,min 40, max 300)
```

The capabilities are acknowledged for both call legs.

```
*Mar  1 15:36:05.028: //45/xxxxxxxxxxxx/CCAPI/cc_api_caps_ack: (dstVdbPtr=0x637EC1E0,
dstCallId=0x2C, srcCallId=0x2D,
    caps={codec=0x4, fax_rate=0x2, vad=0x2, modem=0x0
           codec_bytes=20, signal_type=2, seq_num_start=2944})
*Mar  1 15:36:05.028: //44/xxxxxxxxxxxx/CCAPI/cc_api_caps_ack: (dstVdbPtr=0x62EC
61A4, dstCallId=0x2D, srcCallId=0x2C,
     caps={codec=0x4, fax_rate=0x2, vad=0x2, modem=0x0
           codec_bytes=20, signal_type=2, seq_num_start=2944})

*Mar  1 15:36:05.032: //44/xxxxxxxxxxxx/CCAPI/cc_api_voice_mode_event: callID=0x2C
*Mar  1 15:36:05.032: //44/45F2AAE28044/CCAPI/cc_api_voice_mode_event: Call Pointer
=634A430C
*Mar  1 15:36:05.032: //44/xxxxxxxxxxxx/SSAPP:-1:-1/sess_appl:
ev(52=CC_EV_VOICE_MODE_DONE), cid(44), disp(0)
*Mar  1 15:36:05.032: //44/45F2AAE28044/SSAPP:10002:21/ssaTraceSct:
Router#
Router#cid(44)st(SSA_CS_ACTIVE)ev(SSA_EV_VOICE_MODE_DONE)oldst(SSA_CS_CONFERENCING)cfid(21
)csize(2)in(1)fDest(1)
*Mar  1 15:36:05.032: //44/45F2AAE28044/SSAPP:10002:21/ssaTraceSct:
-cid2(45)st2(SSA_CS_ACTIVE)oldst2(SSA_CS_ALERT_RCVD)
*Mar  1 15:36:05.032: //44/45F2AAE28044/SSAPP:10002:21/ssaIgnore: cid(44),
st(SSA_CS_ACTIVE),oldst(5), ev(52)
Router#
Router#! digit punched
Router#
```

The phone at the terminating gateway enters digit 1.

```
*Mar  1 15:36:11.204: //45/xxxxxxxxxxxx/CCAPI/cc_api_call_digit_begin: (dstVdbPt
r=0x637EC1E0, dstCallId=0x2C, srcCallId=0x2D,
    digit=1, digit_begin_flags=0x0, rtp_timestamp=0x0
    rtp_expiration=0x0, dest_mask=0x2)
*Mar  1 15:36:11.504: //45/xxxxxxxxxxxx/CCAPI/cc_api_call_digit_end:
(dstVdbPtr=0x637EC1E0, dstCallId=0x2C, srcCallId=0x2D,
    digit=1,duration=300,xruleCallingTag=0,xruleCalledTag=0, dest_mask=0x2),
digit_tone_mode=0
```

The phone at the terminating gateway enters digit 2.

```
*Mar  1 15:36:11.604: //45/xxxxxxxxxxxx/CCAPI/cc_api_call_digit_begin:
(dstVdbPtr=0x637EC1E0, dstCallId=0x2C, srcCallId=0x2D,
    digit=2, digit_begin_flags=0x0, rtp_timestamp=0x0
    rtp_expiration=0x0, dest_mask=0x2)
*Mar  1 15:36:11.904: //45/xxxxxxxxxxxx/CCAPI/cc_api_call_digit_end: (dstVdbPtr=
0x637EC1E0, dstCallId=0x2C, srcCallId=0x2D,
    digit=2,duration=300,xruleCallingTag=0,xruleCalledTag=0, dest_mask=0x2),
digit_tone_mode=0
Router#
```

```
*Mar  1 15:36:14.476: //-1/xxxxxxxxxxxx/CCAPI/cc_handle_periodic_timer: Calling the
callback, ccTimerctx - 0x628B6330
*Mar  1 15:36:14.476: //-1/xxxxxxxxxxxx/CCAPI/ccTimerStart: ccTimerctx - 0x628B6330
Router#
Router#!call hung up  The user at the terminating gateway hangs up the call.
Router#
```

The voice gateway receives a disconnect message from the terminating gateway. The cause code is 0x10 which is normal call clearing.

```
*Mar  1 15:36:22.916: //45/xxxxxxxxxxxx/CCAPI/cc_api_call_disconnected: (vdbPtr=
0x62EC61A4, callID=0x2D, cause=0x10)
*Mar  1 15:36:22.920: //45/xxxxxxxxxxxx/SSAPP:-1:-1/sess_appl:
ev(11=CC_EV_CALL_DISCONNECTED), cid(45), disp(0)
*Mar  1 15:36:22.920: //45/45F2AAE28044/SSAPP:0:21/ssaTraceSct:
cid(45)st(SSA_CS_ACTIVE)ev(SSA_EV_CALL_DISCONNECTED)
oldst(SSA_CS_ALERT_RCVD)cfid(21)csize(2)in(0)fDest(0)
*Mar  1 15:36:22.920: //45/45F2AAE28044/SSAPP:0:21/ssaTraceSct:
-cid2(44)st2(SSA_CS_ACTIVE)oldst2(SSA_CS_ACTIVE)
*Mar  1 15:36:22.920: ssa: Disconnected cid(45) state(5) cause(0x10)
```

The voice gateway begins tearing down the conference and dropping the bridge.

```
*Mar  1 15:36:22.920: //-1/xxxxxxxxxxxx/CCAPI/ccConferenceDestroy: (confID=0x15, tag=0x0)
*Mar  1 15:36:22.920: //45/xxxxxxxxxxxx/CCAPI/cc_api_bridge_drop_done: (confID=0x15,
srcIF=0x62EC61A4, srcCallID=0x2D, dstCallID=0x2C, disposition=0 tag=0x0)
*Mar  1 15:36:22.920: //44/xxxxxxxxxxxx/CCAPI/cc_api_bridge_drop_done: (confID=0x15,
srcIF=0x637EC1E0, srcCallID=0x2C, dstCallID=0x2D, disposition=0 tag=0x0)
*Mar  1 15:36:22.924: //44/xxxxxxxxxxxx/SSAPP:-1:-1/sess_appl:
ev(30=CC_EV_CONF_DESTROY_DONE), cid(44), disp(0)
*Mar  1 15:36:22.924: //44/45F2AAE28044/SSAPP:10002:21/ssaTraceSct:
cid(44)st(SSA_CS_CONF_DESTROYING)ev(SSA_EV_CONF_DESTROY_DONE)
oldst(SSA_CS_ACTIVE)cfid(21)csize(2)in(1)fDest(1)
*Mar  1 15:36:22.924: //44/45F2AAE28044/SSAPP:10002:21/ssaTraceSct: -cid2(45)st2
(SSA_CS_CONF_DESTROYING)oldst2(SSA_CS_ACTIVE)
*Mar  1 15:36:22.924: //45/45F2AAE28044/SSAPP:0:-1/ssaConfDestroyDone:
*Mar  1 15:36:22.924: //44/xxxxxxxxxxxx/CCAPI/ccCallDisconnect: (callID=0x2C, cause=0x10
tag=0x0)
```

The voice gateway stops call accounting on the incoming call, indicated by the leg_type=True message. The cause code is then set for the originating leg.

```
*Mar  1 15:36:22.924: //44/45F2AAE28044/CCAPI/ccCallDisconnect: calling accounting start
for callID=44 leg_type=1
*Mar  1 15:36:22.924: //44/45F2AAE28044/CCAPI/ccCallDisconnect: existing_cause = 0x0,
new_cause = 0x10
*Mar  1 15:36:22.924: //44/xxxxxxxxxxxx/CCAPI/cc_api_get_transfer_info: (callID=0x2C)
*Mar  1 15:36:22.924: //45/xxxxxxxxxxxx/CCAPI/ccCallDisconnect: (callID=0x2D, cause=0x10
tag=0x0)
```

The voice gateway stops call accounting for the outgoing call, indicated by the leg_type=False message. The cause code is verified for the terminating leg.

```
*Mar  1 15:36:22.924: //45/45F2AAE28044/CCAPI/ccCallDisconnect: calling accounting start
for callID=45 leg_type=0
*Mar  1 15:36:22.924: //45/45F2AAE28044/CCAPI/ccCallDisconnect: existing_cause = 0x10,
new_cause = 0x10
*Mar  1 15:36:22.924: //45/45F2AAE28044/CCAPI/ccCallDisconnect: using the existing_cause
0x10
*Mar  1 15:36:22.928: //45/xxxxxxxxxxxx/CCAPI/cc_api_get_transfer_info: (callID=0x2D)
*Mar  1 15:36:22.932: //-1/xxxxxxxxxxxx/CCAPI/cc_api_icpif: expect factor = 0
*Mar  1 15:36:22.932: //-1/xxxxxxxxxxxx/CCAPI/g113_calculate_impairment: (delay=79,
      loss=0), Io=0 Iq=0 Idte=0 Idd=0 Ie=10 Itot=10
```

```
*Mar  1 15:36:22.932: //-1/xxxxxxxxxxxx/CCAPI/cc_decr_if_call_volume: the remote IP is
171.69.85.111
*Mar  1 15:36:22.932: //-1/xxxxxxxxxxxx/CCAPI/cc_decr_if_call_volume: hwidb is Ethernet0/0
*Mar  1 15:36:22.932: //-1/xxxxxxxxxxxx/CCAPI/cc_decr_if_call_volume: reduce callnum of
entry: 0, voip: 0, mmoip: 0
*Mar  1 15:36:22.932: //-1/xxxxxxxxxxxx/CCAPI/cc_decr_if_call_volume: remove anentry
*Mar  1 15:36:22.932: //45/xxxxxxxxxxxx/CCAPI/cc_api_call_disconnect_done:
(vdbPtr=0x62EC61A4, callID=0x2D, disp=0, tag=0x0)
*Mar  1 15:36:22.932: //45/45F2AAE28044/CCAPI/ccTDPvtProfileTableObjectAccessManager: No
profileTable set for callID[45]
*Mar  1 15:36:22.936: //45/xxxxxxxxxxxx/CCAPI/ccTDUtilGetDataByRef: No tdObjectfound in
profileTable for tagID[6] of callID[45]
*Mar  1 15:36:22.936: //45/45F2AAE28044/CCAPI/cc_delete_call_entry: not incoming entry
*Mar  1 15:36:22.936: //45/45F2AAE28044/CCAPI/cc_delete_call_entry: entry's incoming
FALSE.
*Mar  1 15:36:22.936: //45/45F2AAE28044/CCAPI/cc_delete_call_entry: is_incomingis FALSE
*Mar  1 15:36:22.940: //45/xxxxxxxxxxxx/SSAPP:-1:-1/sess_appl:
ev(12=CC_EV_CALL_DISCONNECT_DONE), cid(45), disp(0)
*Mar  1 15:36:22.940: //45/45F2AAE28044/SSAPP:0:-1/ssaTraceSct: cid(45)st(SSA_CS
_DISCONNECTING)ev(SSA_EV_CALL_DISCONNECT_DONE)oldst(SSA_CS_ACTIVE)cfid(-1)csize(2)in(0)fDe
st(0)
*Mar  1 15:36:22.940: //45/45F2AAE28044/SSAPP:0:-1/ssaTraceSct:
-cid2(44)st2(SSA_CS_DISCONNECTING)oldst2(SSA_CS_CONF_DESTROYING)
*Mar  1 15:36:22.940: //45/45F2AAE28044/SSAPP:0:-1/ssaDisconnectDone:
*Mar  1 15:36:22.940: //45/45F2AAE28044/SSAPP:0:-1/ssaAAA_CheckAccounting: accounting
generation enabled
*Mar  1 15:36:22.940: //45/xxxxxxxxxxxx/CCAPI/ccCallSetAAA_Accounting: callID=0x2D,
accounting=0
*Mar  1 15:36:22.944: //-1/xxxxxxxxxxxx/CCAPI/cc_decr_if_call_volume: not the VoIP or
MMoIP
*Mar  1 15:36:22.948: //44/xxxxxxxxxxxx/CCAPI/cc_api_call_disconnect_done:
(vdbPtr=0x637EC1E0, callID=0x2C, disp=0, tag=0x0)
*Mar  1 15:36:22.948: //44/45F2AAE28044/CCAPI/cc_delete_call_entry:
ccFreeRawMsgInfo(0x6307595C)
*Mar  1 15:36:22.948: //44/45F2AAE28044/CCAPI/cc_delete_call_entry: Decrement call volume
counter 1
*Mar  1 15:36:22.948: //44/45F2AAE28044/CCAPI/cc_delete_call_entry: current call volume: 0
*Mar  1 15:36:22.948: //44/45F2AAE28044/CCAPI/cc_delete_call_entry: entry's incoming TRUE.
*Mar  1 15:36:22.948: //44/45F2AAE28044/CCAPI/cc_delete_call_entry: is_incomingis TRUE
*Mar  1 15:36:22.948: //44/45F2AAE28044/CCAPI/cc_delete_call_entry: Deleting
profileTable[0x6380E11C]
*Mar  1 15:36:22.948: //-1/xxxxxxxxxxxx/CCAPI/ccTDDestructTDHashProfileTab: Destructor
Profile Table (0x6380E11C)
*Mar  1 15:36:22.948: //-1/xxxxxxxxxxxx/CCAPI/ccTDDestructInstanceTDObject:
tdObject[0x63401148] tagID[5]
*Mar  1 15:36:22.948: //-1/xxxxxxxxxxxx/CCAPI/ccTDDestructInstanceTDObject:
tdObject[0x638BC1AC] tagID[6]
*Mar  1 15:36:22.956: //44/xxxxxxxxxxxx/SSAPP:-1:-1/sess_appl:
ev(12=CC_EV_CALL_DISCONNECT_DONE), cid(44), disp(0)
*Mar  1 15:36:22.956: //44/45F2AAE28044/SSAPP:10002:-1/ssaTraceSct:
cid(44)st(SSA_CS_DISCONNECTING)ev(SSA_EV_CALL_DISCONNECT_DONE)oldst(SSA_CS_CONF_DESTROYING
)cfid(-1)csize(1)in(1)fDest(1)
Router#
*Mar  1 15:36:22.956: //44/45F2AAE28044/SSAPP:10002:-1/ssaDisconnectDone:
```