



# Global Call E1/T1 CAS/R2

## Technology Guide

---

*April 2006*



INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT. Intel products are not intended for use in medical, life saving, life sustaining, critical control or safety systems, or nuclear facility applications.

Intel may make changes to specifications and product descriptions at any time, without notice.

This Global Call E1/T1 CAS/R2 Technology Guide as well as the software described in it is furnished under license and may only be used or copied in accordance with the terms of the license. The information in this manual is furnished for informational use only, is subject to change without notice, and should not be construed as a commitment by Intel Corporation. Intel Corporation assumes no responsibility or liability for any errors or inaccuracies that may appear in this document or any software that may be provided in association with this document.

Except as permitted by such license, no part of this document may be reproduced, stored in a retrieval system, or transmitted in any form or by any means without the express written consent of Intel Corporation.

Copyright © 2005-2006, Intel Corporation

Dialogic, Intel, Intel logo, and Intel NetStructure are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

\* Other names and brands may be claimed as the property of others.

Publication Date: April 2006

Document Number: 05-2445-002

Intel  
1515 Route 10  
Parsippany, NJ 07054

For **Technical Support**, visit the Intel Telecom Support Resources website at:  
<http://developer.intel.com/design/telecom/support>

For **Products and Services Information**, visit the Intel Telecom and Compute Products website at:  
<http://www.intel.com/design/network/products/telecom>

For **Sales Offices** and other contact information, visit the Buy Telecom Products page at:  
<http://www.intel.com/buy/networking/telecom.htm>

# Contents

---

	<b>Revision History</b> .....	9
	<b>About This Publication</b> .....	11
<b>1</b>	<b>E1/T1 CAS/R2 Overview</b> .....	13
1.1	Making Telephone Calls: Transmission of Digits and Signaling Information .....	13
1.2	Making Long Distance and Global Telephone Calls .....	15
1.3	T1 Robbed Bit Signaling Concepts .....	15
1.4	E1 CAS Signaling Concepts .....	16
1.5	R2MF Signaling Concepts .....	17
1.5.1	R2MF Multifrequency Combinations .....	18
1.5.2	R2MF Signal Meanings .....	18
1.5.3	R2MF Compelled Signaling .....	19
1.6	Direct Dialing In (DDI) Service .....	20
<b>2</b>	<b>Global Call Architecture for E1/T1 CAS/R2</b> .....	21
<b>3</b>	<b>E1/T1 CAS/R2 Call Scenarios</b> .....	23
<b>4</b>	<b>E1/T1 CAS/R2-Specific Operations</b> .....	25
4.1	Call Progress and Call Analysis .....	25
4.1.1	Call Analysis with DM3 Boards .....	26
4.1.2	Call Analysis with Springware Boards .....	28
4.1.3	Call Analysis Functionality for PDK Protocols .....	29
4.1.4	Tone Definitions for PDK Protocols .....	32
4.1.5	Call Analysis Functionality for ICAPI Protocols .....	35
4.2	CAS Pattern Signal Declarations .....	37
4.2.1	CAS_SIGNAL_TRANS_t .....	37
4.2.2	CAS_SIGNAL_PULSE_t .....	38
4.2.3	CAS_SIGNAL_TRAIN_t .....	39
4.3	Dynamic Trunk Configuration .....	40
4.3.1	Setting the Line Type and Coding for a Trunk .....	41
4.3.2	Specifying the Protocol for a Trunk .....	43
4.4	Resource Association .....	44
4.5	Resource Allocation and Routing .....	45
4.5.1	Dedicated Voice Resources .....	45
4.5.2	Shared Voice Resources .....	47
4.6	Alarm Handling .....	49
4.6.1	Alarm Handling for DM3 boards .....	49
4.6.2	Alarm Handling for Springware Boards .....	52
4.7	Run Time Configuration of the PDKRT Call Control Library .....	56
4.8	Run Time Configuration of PDK Protocol Parameters .....	57
4.9	Determining the Protocol Version .....	59
<b>5</b>	<b>E1/T1 CAS/R2 Protocols</b> .....	61
5.1	Protocols Supported .....	61

5.2	Protocol File Naming Conventions . . . . .	61
5.3	Protocol Components . . . . .	63
5.3.1	Protocol Modules . . . . .	63
5.3.2	Country Dependent Parameter (.cdp) Files . . . . .	64
<b>6</b>	<b>Building Global Call E1/T1 CAS/R2 Applications . . . . .</b>	<b>65</b>
6.1	Header Files . . . . .	65
6.2	Required Libraries . . . . .	65
6.3	Required System Software . . . . .	65
<b>7</b>	<b>Debugging Global Call E1/T1 CAS/R2 Applications . . . . .</b>	<b>67</b>
7.1	Introduction . . . . .	67
7.2	Debugging Applications that Use PDK Protocols . . . . .	67
7.2.1	Enabling and Disabling the Logging . . . . .	67
7.2.2	Populating and Using a CCLIB_START_STRUCT . . . . .	68
7.2.3	Defining the GC_PDK_START_LOG Environment Variable . . . . .	72
7.3	Debugging Applications that Use ICAP1 Protocols . . . . .	72
<b>8</b>	<b>E1/T1 CAS/R2-Specific Function Information . . . . .</b>	<b>75</b>
8.1	Global Call Functions Supported by E1/T1 CAS/R2 . . . . .	75
8.2	Global Call Function Variances for E1/T1 CAS/R2 . . . . .	82
8.2.1	gc_AcceptCall( ) Variances for E1/T1 CAS/R2 . . . . .	83
8.2.2	gc_AnswerCall( ) Variances for E1/T1 CAS/R2 . . . . .	83
8.2.3	gc_BlindTransfer( ) Variances for E1/T1 CAS/R2 . . . . .	83
8.2.4	gc_CallAck( ) Variances for E1/T1 CAS/R2 . . . . .	84
8.2.5	gc_Close( ) Variances for E1/T1 CAS/R2 . . . . .	84
8.2.6	gc_CompleteTransfer( ) Variances for E1/T1 CAS/R2 . . . . .	85
8.2.7	gc_Detach( ) Variances for E1/T1 CAS/R2 . . . . .	85
8.2.8	gc_DropCall( ) Variances for E1/T1 CAS/R2 . . . . .	85
8.2.9	gc_Extension( ) Variances for E1/T1 CAS/R2 . . . . .	86
8.2.10	gc_GetCallInfo( ) Variances for E1/T1 CAS/R2 . . . . .	86
8.2.11	gc_GetParm( ) Variances for E1/T1 CAS/R2 . . . . .	87
8.2.12	gc_HoldCall( ) Variances for E1/T1 CAS/R2 . . . . .	87
8.2.13	gc_MakeCall( ) Variances for E1/T1 CAS/R2 . . . . .	88
8.2.14	gc_OpenEx( ) Variances for E1/T1 CAS/R2 . . . . .	89
8.2.15	gc_ResetLineDev( ) Variances for E1/T1 CAS/R2 . . . . .	91
8.2.16	gc_RetrieveCall( ) Variances for E1/T1 CAS/R2 . . . . .	92
8.2.17	gc_SetBilling( ) Variances for E1/T1 CAS/R2 . . . . .	92
8.2.18	gc_SetChanState( ) Variances for E1/T1 CAS/R2 . . . . .	92
8.2.19	gc_SetEvtMsk( ) Variances for E1/T1 CAS/R2 . . . . .	93
8.2.20	gc_SetParm( ) Variances for E1/T1 CAS/R2 . . . . .	93
8.2.21	gc_SetupTransfer( ) Variances for E1/T1 CAS/R2 . . . . .	94
8.2.22	gc_Start( ) and gc_Stop( ) Variances for E1/T1 CAS/R2 . . . . .	94
8.2.23	gc_StartTrace( ) Variances for E1/T1 CAS/R2 . . . . .	94
8.2.24	gc_SwapHold( ) Variances for E1/T1 CAS/R2 . . . . .	94
<b>9</b>	<b>E1/T1 CAS/R2-Specific Data Structures . . . . .</b>	<b>97</b>
<b>10</b>	<b>E1/T1 CAS/R2-Specific Event Cause Values . . . . .</b>	<b>101</b>
<b>11</b>	<b>Supplementary Reference Information . . . . .</b>	<b>105</b>



<b>Index .....</b>	<b>107</b>
--------------------	------------

## ***Figures***

---

1	Global Call Architecture When Using E1/T1 CAS/R2. ....	22
---	--	----

# Tables

---

1	Signaling Used to Dial (Hz) . . . . .	14
2	Global Call Call Progress Settings. . . . .	27
3	Call Analysis Support on DM3 Boards with CAS. . . . .	27
4	TONE_t Signal Definition Parameters . . . . .	33
5	CAS_SIGNAL_TRANS_t Signal Definition Parameters . . . . .	38
6	CAS_SIGNAL_PULSE_t Signal Definition Parameters. . . . .	39
7	CAS_SIGNAL_TRAIN_t Signal Definition Parameters . . . . .	40
8	Alarms That Can Be Transmitted on E1 and T1 Interfaces on DM3 Boards. . . . .	50
9	Alarms That Can Be Transmitted on E1 and T1 Interfaces on Springware Boards . . . . .	53
10	Configurable PDKRT Call Control Library Parameters . . . . .	56
11	PSL and SYS Parameters . . . . .	58
12	Configurable PDK Protocol Parameters . . . . .	58
13	Protocol File Naming Conventions . . . . .	62
14	Sample ICAPI Protocol File Set. . . . .	63
15	Sample PDK Protocol File Set. . . . .	63
16	cclib_data Fields and Values. . . . .	69
17	Loglevel Parameter Values . . . . .	70
18	Service Parameter Values. . . . .	70
19	Cachedump Parameter Values . . . . .	71
20	Sample Channel Parameter Values . . . . .	71
21	icapi.cfg File Parameters . . . . .	73
22	Parameters Supported, gc_GetParm( ) and gc_SetParm( ) . . . . .	93
23	Call Control Library Cause Values When Using DM3 Boards. . . . .	101
24	Firmware-Related Cause Values When Using DM3 Boards. . . . .	101







## Revision History

---

This revision history summarizes the changes made in each published version of this document.

Document No.	Publication Date	Description of Revisions
05-2445-002	April 2006	<p><a href="#">Global Call Architecture for E1/T1 CAS/R2</a> chapter: Clarified which call control libraries are applicable to DM3 boards and to Springware boards.</p> <p><a href="#">E1/T1 CAS/R2-Specific Operations</a> chapter: In <a href="#">Dynamic Trunk Configuration</a> section, added note clarifying the use of <b>gc_ResetLineDev( )</b> before <b>gc_SetConfigData( )</b> when performing dynamic trunk configuration.</p> <p>Updated <a href="#">Alarm Handling for DM3 boards</a> section to more accurately specify the alarms that can be transmitted to the remote side and provide a mapping to the 0x1626 parameter in the CONFIG file, which is used for trunk preconditioning.</p> <p>Updated <a href="#">Alarm Handling for Springware Boards</a> section to more accurately specify the alarms that can be transmitted to the remote side.</p> <p><a href="#">E1/T1 CAS/R2-Specific Function Information</a> chapter: In <a href="#">gc_SetChanState( ) Variances for E1/T1 CAS/R2</a> section, added note about protocols that do not send the blocking pattern by default (PTR 36726).</p> <p>In <a href="#">gc_SetupTransfer( ) Variances for E1/T1 CAS/R2</a> section, corrected the name of the <b>gc_SetupTransfer( )</b> function; the “u” is lower case, not upper case (PTR 35811).</p>
05-2445-001	July 2005	<p>Initial version of document. Much of the information contained in this document was previously published in the <i>Global Call E1/T1 CAS/R2 Technology User's Guide for Linux and Windows</i>, document number 05-0615-011.</p>





## About This Publication

---

The following topics provide information about this publication:

- [Purpose](#)
- [Intended Audience](#)
- [How to Use This Publication](#)
- [Related Information](#)

### Purpose

This guide is for users of the Global Call API writing applications that use E1/T1 CAS/R2 technology. This guide provides Global Call E1/T1 CAS/R2 specific information only and should be used in conjunction with the *Global Call API Programming Guide* and the *Global Call API Library Reference*, which describe the generic behavior of the Global Call API.

### Intended Audience

This guide is intended for:

- Distributors
- System Integrators
- Toolkit Developers
- Independent Software Vendors (ISVs)
- Value Added Resellers (VARs)
- Original Equipment Manufacturers (OEMs)

This publication assumes that the audience is familiar with the Windows\* and Linux\* operating systems and has experience using the C programming language.

### How to Use This Publication

This guide is divided into the following chapters:

- [Chapter 1, “E1/T1 CAS/R2 Overview”](#) gives a brief introduction to E1/T1 CAS/R2 concepts for novice users.
- [Chapter 2, “Global Call Architecture for E1/T1 CAS/R2”](#) provides an overview of the Global Call software architecture when using E1/T1 CAS/R2 technology.
- [Chapter 3, “E1/T1 CAS/R2 Call Scenarios”](#) discusses call scenarios.

- [Chapter 4, “E1/T1 CAS/R2-Specific Operations”](#) describes how to use the Global Call API to perform E1/T1 CAS/R2 specific operations, such call progress and call analysis, resource association, and others.
- [Chapter 5, “E1/T1 CAS/R2 Protocols”](#) describes the protocol conventions used and programming considerations when incorporating individual country protocol(s) into your application. (More detailed information about each protocol appears in the *Global Call Country Dependent Parameters (CDP) for PDK Protocols Configuration Guide*.)
- [Chapter 6, “Building Global Call E1/T1 CAS/R2 Applications”](#) describes the E1/T1 CAS/R2 specific header files and libraries required when building applications.
- [Chapter 7, “Debugging Global Call E1/T1 CAS/R2 Applications”](#) provides information for debugging Global Call applications that use E1/T1 CAS/R2 technology.
- [Chapter 8, “E1/T1 CAS/R2-Specific Function Information”](#) describes the additional functionality of specific Global Call functions used with E1/T1 CAS/R2 technology.
- [Chapter 11, “Supplementary Reference Information”](#) lists references to publications about E1/T1 CAS/R2 technology.

## Related Information

Refer to the following documents and web sites for more information about developing applications that use the Global Call API:

- *Global Call API Programming Guide*
- *Global Call API Library Reference*
- *Global Call Country Dependent Parameters (CDP) for PDK Protocols Configuration Guide*
- <http://developer.intel.com/design/telecom/support> (for technical support)
- <http://www.intel.com/design/network/products/telecom> (for product information)

This chapter provides overview information about the following topics:

- Making Telephone Calls: Transmission of Digits and Signaling Information . . . . . 13
- Making Long Distance and Global Telephone Calls. . . . . 15
- T1 Robbed Bit Signaling Concepts . . . . . 15
- E1 CAS Signaling Concepts . . . . . 16
- R2MF Signaling Concepts. . . . . 17
- Direct Dialing In (DDI) Service . . . . . 20

## 1.1 Making Telephone Calls: Transmission of Digits and Signaling Information

Historically, making a telephone call started with taking your telephone handset out of its cradle. This action caused your telephone to go off-hook. For analog telephones, going off-hook closes a circuit (called the local loop) connected to the local Central Office (CO) and causes a loop current to flow through the local loop circuit created.

The CO reacts by generating dial tone (typically, a combination of 350 Hz and 440 Hz tones), which indicates that you can dial. Traditionally, you would dial your number using pulse dialing (also called rotary dialing). Pulse dialing sends digit information to the CO by momentarily opening and closing (or breaking) the local loop from the calling party to the CO. This local loop is broken once for the digit 1, twice for 2, etc., and 10 times for the digit 0. As each number is dialed, the loop current is switched on and off, resulting in a number of pulses being sent to your local CO.

Alternatively, you may dial a number using tone dialing, wherein sounds represent the digits dialed (0 through 9, # and \* are dialing digits). Each digit is assigned a unique pair of frequencies called Dual Tone Multi Frequency (DTMF) digits (see Table 1). Although DTMF signaling is designed for operation on international networks with 15 multifrequency combinations in each direction, in national networks it can be used with a reduced number of signaling frequencies (for example, 10 multifrequency combinations).

In addition to the DTMF digit standard, telcos also use a Multi Frequency (MF) digit standard (see Table 1). MF digits are typically used for CO-to-CO signaling. The MF digit standard is similar to the DTMF digit standard except that different pairs of frequencies are assigned. Some MF digits use approximately the same frequencies as DTMF digits; for example, the digit 4 uses 770 and 1209 Hz for DTMF transmissions or 700 and 1300 Hz for MF transmissions. Because of this frequency overlap, MF digits could be mistaken for DTMF digits if the incorrect tone detection is enabled.

The accuracy of digit detection depends on:

- the digit sent
- the type of detection, MF or DTMF, enabled when the digit is detected. See the *Voice API Library Reference* for details.

**Table 1. Signaling Used to Dial (Hz)**

Code	Pulse	DTMF	MF	R2MF Forward	R2MF Backward
1	1	697, 1209	700, 900	1380, 1500	1140, 1020
2	2	697, 1336	700, 1100	1380, 1620	1140, 900
3	3	697, 1477	900, 1100	1500, 1620	1020, 900
4	4	770, 1209	700, 1300	1380, 1740	1140, 780
5	5	770, 1336	900, 1300	1500, 1740	1020, 780
6	6	770, 1477	1100, 1300	1620, 1740	900, 780
7	7	852, 1209	700, 1500	1380, 1860	1140, 660
8	8	852, 1336	900, 1500	1500, 1860	1020, 660
9	9	852, 1477	1100, 1500	1620, 1860	900, 660
0	10	941, 1336	1300, 1500	1740, 1860	780, 660
*	-	941, 1209	1100, 1700	1380, 1980	1140, 540
#	-	941, 1477	1500, 1700	1500, 1980	1020, 540

For each call, signaling information (off-hook, number dialed) must be detected by the local CO and then sent to each successive CO until the destination CO is reached. The destination CO attempts to connect to the called party. Concurrently, the destination CO sends back signaling information (such as line busy, network busy signals, etc.) representing the condition or status of the called party's line. This signaling information passes through the network as audio tones or as signaling bits. The number of tones used and the frequency combinations used to convey this signaling information vary from country to country and from telco to telco. In addition, private networks may combine various signaling techniques.

After dialing, you listen to hear the progress and status of the call:

- Ringing tones (ringback) indicate that ring voltage has been applied to the called party's line.
- A busy tone is heard when the called party's telephone is off-hook.
- A fast busy tone may be heard if the telephone network is busy.
- An operator intercept signal is heard if an invalid number is dialed. The operator intercept signal is three rising tones followed by a recording.

**Note:** No ringing tones are heard when connected to some telcos.

The CO typically indicates the progress of making a call by generating these various tones. When making long distance calls, the telco may make brief drops in loop current to indicate:

- an acknowledgment that the distant CO was reached
- that the calling party's line went off-hook

After a call is connected, a telco service may be requested by a flash-hook. A flash-hook puts the telephone on-hook briefly, long enough for the CO to detect the flash-hook, but not long enough to cause a disconnect. A flash-hook may signal a request for a second dial tone to allow 3-way conferencing or to transfer the call.

At the completion of the call, one or both parties hang up the telephone. Typically, the CO sends a disconnect signal. However, some telcos don't send a disconnect signal; therefore a local CO must use other methods to detect a remote disconnect.

## 1.2 Making Long Distance and Global Telephone Calls

Long distance calls may involve transmitting dialing and other signaling information from the local CO, through several intermediate COs, to the distant called party's CO and then connecting to the called party. A mixture of signaling systems and protocols may be encountered, especially when making global calls. Local call signaling must be translated into signaling that may pass over analog lines, T1 digital trunks, E1 digital trunks, optical fiber, satellite links, etc. All signaling sent over digital trunks must be converted to bits that can be transmitted or multiplexed with the digitized voice transmissions.

Each telco, country, or region tends to apply different signaling standards that must be observed to ensure that a call gets switched through to the called party. For example, some telcos may encode E&M (Ear and Mouth) signals onto the voice path using a single frequency (SF) tone. When present, this tone indicates an on-hook condition. Otherwise, the line is considered to be off-hook (absence of tone). Typically, when the same manufacturer's product is connected to both ends of a digital trunk, then the signaling technique used is transparent as long as all signaling is handled.

## 1.3 T1 Robbed Bit Signaling Concepts

A T1 trunk operates at 1.544 Mbps divided into 24 time slots with each time slot operating at 64 kbps [digital signal level 1 (DS-1) rate]. A single 8-bit sample from each of 24 voice channels comprises a D4 frame of 24 time slots on a T1 trunk. Twelve D4 frames make up a D4 superframe.

Signaling information is carried on a T1 trunk by two signaling bits, an A-bit and a B-bit. Each time slot in the sixth frame of a D4 superframe has the least significant bit replaced with A-bit signaling information. Likewise, each time slot in the twelfth frame of the D4 superframe has the least significant bit replaced with B-bit signaling information. This method of replacing the least significant bit with signaling information is called robbed bit signaling. Thus, a T1 robbed bit trunk carries all signaling within the voice time slot (channel) itself.

Dialing, if not done using DTMF or MF tones, is accomplished by alternating the A and B signaling bits between 0 and 1 to mimic rotary dial pulses. Signaling bits represent the state of the M lead on the E&M interface of the calling party. When the called party answers, the M lead returns continuous 1s. When a party hangs up, their signal bits revert to 0s to indicate on-hook. Some telcos invert these signaling bits so that 0 = off-hook and 1 = on-hook.

New telco services may require the use of more than the four signaling states provided by the A and B bits. An extended superframe (ESF) adopted by AT&T provides two additional signaling bits, the C-bit in frame 18 and the D-bit in frame 24.

## 1.4 E1 CAS Signaling Concepts

An E1 digital trunk operates at 2.048 Mbps divided into 32 time slots with each time slot operating at 64 kbps. These 32 time slots include:

- 30 time slots available for up to 30 voice calls
- one time slot dedicated to carrying frame synchronization information (time slot 0)
- one time slot dedicated to carrying signaling information (time slot 16)

With this method of signaling, each traffic channel has a dedicated signaling channel, that is, channel associated signaling (CAS). The signaling for a particular traffic circuit is permanently associated with that circuit. For E1 CAS, the signaling channel for each traffic channel is located in time slot 16, which is multiplexed between all 30 traffic channels.

E1 CAS service is available in Europe, Africa, Australia, and in parts of Asia and South America. The Conference des Administrations Europeenes des Postes et Telecommunications (CEPT) defines how a PCM carrier system in E1 areas will be used. In addition, the E1 CAS service may carry national and international signaling bits set in time slot 0:

- The international bit occupies the most significant bit (bit position 7) in time slot 0 of each frame.
- The national bits occupy bit positions 0 through 4 of time slot 0 of every second frame.

For each E1 CAS call, signaling information is sent to the local CO and then to each successive CO until the destination CO is reached. The destination CO attempts to connect to the called party. Concurrently, the destination CO sends back signaling information representing the condition or status of the called party's line. This signaling information passes through the network as audio tones. R2MF signaling is the international standard for conveying call status using these audio tones. However, the number of tones used, the frequency combinations used, and the adherence to the R2 standard can vary from country to country.

Also, whenever a call is switched via networks or protocols that do not support full R2MF signaling, call information may be lost. Although many protocols do not require call analysis because the called party condition is received via R2 tones, when operating in environments where call information may be lost, call progress tones (busy, ringback, SIT tones, etc.) may be useful in determining the condition of a call.

The following sections describe R2MF signaling as it is used in a full R2 network, global tone detection considerations, and Global Call call analysis capability. The protocols do not require call analysis because the called party condition is received via R2 tones; but you can modify the country dependent parameters (.cdp) file so that the protocol can use either call progress tones or call analysis.



## 1.5 R2MF Signaling Concepts

R2MF signaling is an international signaling system on E1 that transmits numeric and other information relating to the called and the calling subscribers' lines. R2MF signals that control the call setup are referred to as interregister signals.

For each call, whether an inbound or an outbound call, the entity making the call is the "calling party" and the entity receiving the call is the "called party." For an inbound call, the calling party is eventually connected to a central office (CO) that connects to the customer premises equipment (CPE) of the called party. For this inbound call, the CO is referred to as the outgoing register and the CPE as the incoming register. Signals sent from the CO are forward signals; signals sent from the CPE are backward signals. The outgoing register (CO) sends forward interregister signals and receives backward interregister signals. The incoming register (CPE) receives forward interregister signals and sends backward interregister signals.

For an outbound call, the calling party's CPE connects to the CO that switches the outbound call to the called party. For an outbound call, the signaling described above is reversed. That is, signals sent from the CPE are forward signals and signals sent from the CO are backward signals.

In addition, address signals can provide the telephone number of the called party's line. For national traffic, the address signals can also provide the telephone number of a calling party's line for automatic number identification (ANI) applications.

R2MF signals used for supervisory signaling on the network are called line signals.

For example, a calling party sends the first dialed digits to the local CO. The local CO uses these digits to determine the next CO in the connection chain. The next CO uses these first dialed digits to determine if they are the destination CO or if the call is to be switched to another CO. Eventually, the call reaches the destination CO. At the destination CO, the call is received and acknowledged. The destination CO eventually gets the last dialed digits, which exactly identify the called party.

The destination CO checks the called party's line to determine if it is clear, idle, busy, etc. The destination CO then generates and sends a B-tone backwards to the calling party to indicate the condition of the line. If the called party's line is free, the destination CO applies ringing to the line and sends ringback tones backwards to the calling party. When the called party answers the call, the calling party is switched through to the called party. If the called party's line is busy, or in some other condition, the destination CO sends this information backwards to the calling party via R2 tones. The local CO sends all information received from the destination CO to the calling party. When calls are made in countries that adhere to the full R2 protocol standard (for example, Belgium), the condition of the called party's line is always returned to the calling party.

When traversing networks, protocols, or countries, R2 tonal information can be lost. For example:

- In Italy, for an ICAPI protocol, the calling party would need to use busy tone 103 and ringback tone 105 to determine the condition of the called party's line.
- When the call is switched over a T1 span, the B-tones (also called Group B signals, see [Section 1.5.2, "R2MF Signal Meanings"](#), on page 18) are lost and the condition of the called party's line cannot be detected using R2 tones. In this environment, the application must rely on the call progress tones received to determine the condition of the called party's line.

- In Spain, the network is not a full Socotel backbone; therefore, B-tones defining the condition of the called party's line may or may not be sent backwards to the calling party.

### 1.5.1 R2MF Multifrequency Combinations

R2MF signaling uses a multifrequency code system based on six fundamental frequencies in the forward direction (1380, 1500, 1620, 1740, 1860, and 1980 Hz) and a different set of six frequencies in the backward direction (1140, 1020, 900, 780, 660, and 540 Hz).

Each signal is composed of two of the six frequencies, providing 15 different tone combinations in each direction. Although R2MF signaling is designed for operation on international networks with 15 multifrequency combinations in each direction, in national networks it can be used with a reduced number of signaling frequencies (for example, 10 multifrequency combinations). See the *Voice API Library Reference* for lists of these signal tone pairs.

### 1.5.2 R2MF Signal Meanings

The 15 forward signals are classified into Group I forward signals and Group II forward signals. The 15 backward signals are classified into Group A backward signals and Group B backward signals.

In general, Group I forward signals and Group A backward signals are used to control call setup and to transfer address information between the outgoing register (CO) and the incoming register (CPE). The incoming register can signal the outgoing register to change over to Group II and Group B signaling.

Group II forward signals provide the calling party's category and Group B backward signals provide the condition of the called subscriber's line. Group B signals, also called B-tones, are typically the last tone in the protocol. For example, typically a B-3 tone indicates that the called party's line is busy.

Signaling must always begin with a Group I forward signal followed by a Group A backward signal that serves to acknowledge the signal just received; this Group A backward signal may request additional information. Each signal requires a response from the other party. Each response becomes an acknowledgment of the event and an event to which the other party must respond.

Backward signals serve to indicate certain conditions encountered during call setup or to announce switchover to changed signaling, for example, forward signaling switching over to backward signaling. Changeover to Group II and Group B signaling allows information about the state of the called subscriber's line to be transferred.

The incoming register backward signals can request:

- transmission of address:
  - send next digit
  - send digit previous to last digit
  - send second digit previous to last digit sent
  - send third digit previous to last digit sent

- category of the call (the nature and origin):
  - national or international call
  - operator or subscriber
  - data transmission
  - maintenance or test call
- whether the circuit includes a satellite link
- country code and language for international calls
- information on use of an echo suppressor

The incoming register backward signals can indicate:

- address complete - send category of call
- address complete - put call through
- international, national, or local congestion
- condition of subscriber's line:
  - send SIT to indicate long term unavailability
  - line busy
  - unallocated number
  - line free - charge on answer
  - line free - no charge on answer (only for special destinations)
  - line out of order

The meaning of certain forward multifrequency combinations may also vary depending upon their position in the signaling sequence.

See the *Voice API Library Reference* for more details and definitions of R2MF signals.

### 1.5.3 R2MF Compelled Signaling

Compelled signaling protocols vary from country to country and are grouped into two main categories, both of which are supported by the Global Call software:

- R2MF derived from the CCITT (International Telegraph and Telephone Consultative Committee) standard, where the response tones can carry information from the receiver to the sender. This standard provides a consistent handshake, where the sender always initiates with a forward tone, and the receiver always responds with a backward tone.
- MF Socotel, where the response tone is a standard, single frequency acknowledgment tone that cannot carry additional information. In this standard, the handshake of forward and backward tones changes direction when the receiver needs to send information back to the sender.

The Global Call software provides network device independence by shielding the application from protocol-specific details while giving access to each protocol's full range of features. The compelled signaling feature uses tone generation and detection IDs that are defined at system initialization.

R2MF interregister signaling uses forward and backward compelled signaling. With compelled signaling, each signal is sent until a response (a return) signal is generated. This return signal is sent until responded to by the other party. Each signal stays on until the other party responds, thus compelling a response from the other party. Compelled signaling provides a balance between speed and reliability because it adapts its signaling speed to the working conditions with a minimum loss of reliability.

Compelled signaling must always begin with a Group I forward signal. For an inbound call:

- The CO starts to send the first forward signal.
- As soon as the CPE recognizes this signal, the CPE starts to send a backward signal that serves as an acknowledgment and may also request additional information.
- As soon as the CO recognizes the CPE acknowledging signal, the CO stops sending the forward signal.
- As soon as the CPE recognizes the end of the forward signal, the CPE stops sending the backward signal.
- As soon as the CO recognizes that the CPE stopped sending the backward signal, the CO may start to send the next forward signal.

The above scenario describes the CPE handling of an inbound call. The roles of the CO and the CPE are reversed when the CPE makes an outbound call.

## 1.6 Direct Dialing In (DDI) Service

Since DTMF, MF, and R2MF tone signals can provide the telephone number of the called subscriber's line, these signals may be used by applications providing Direct Dialing In (DDI) service, also called dialed number identification service (DNIS) and analog DNIS for direct inward dialing (DID).

DDI service allows an outside caller to dial an extension within a company without requiring an operator's assistance to transfer the call. The CO passes the last 2, 3, or 4 digits of the dialed number to the CPE, and the CPE completes the call.

# Global Call Architecture for E1/T1 CAS/R2 2

---

This chapter describes the Global Call software architecture when using E1/T1 CAS/R2 technology.

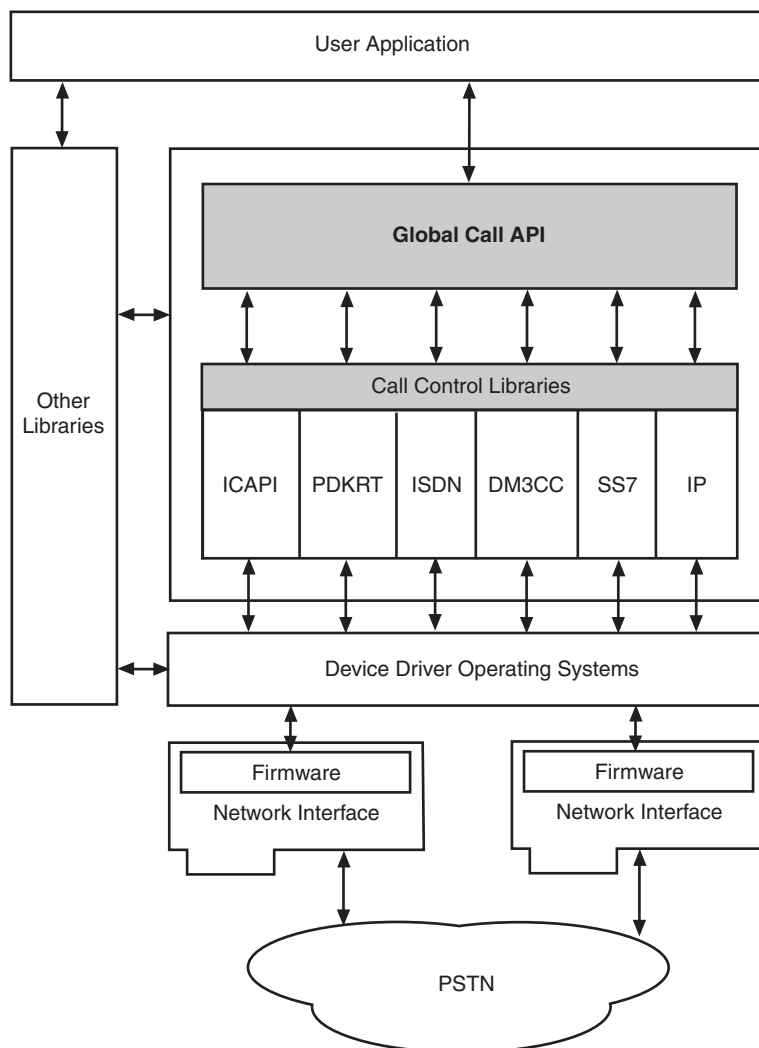
Figure 1 shows the Global Call software architecture with the two key elements from an E1/T1 CAS/R2 viewpoint highlighted:

- The Global Call API is a library of functions that provide primarily call control, but also operation and maintenance functionality to applications.
- The underlying call control libraries provide the interface between the network and the Global Call API library.
  - GC\_DM3CC\_LIB is the DM3CC call control library. This library is used for call control using CAS/R2MF (PDK protocols) signaling on DM3 boards. (It is also used for ISDN on DM3 boards, which is not discussed in this manual; it is covered in the *Global Call ISDN Technology Guide*.)
  - GC\_PDKRT\_LIB is the Protocol Development Kit Run Time (PDKRT) call control library. This library is used for call control using CAS/R2MF (PDK protocols) signaling on Springware boards only.
  - GC\_ICAPI\_LIB is the Interface Control Application Programming Interface (ICAPI) call control library. This library is used for call control using CAS/R2MF (ICAPI protocols) signaling on Springware boards only.

**Note:** The development of the ICAPI protocols supported by Global Call has been capped. Customers should migrate to equivalent protocols developed using the PDK. New protocol development as well as existing protocol support will be on the PDK. ICAPI protocols are supported only on Springware boards. PDK protocols are supported on both DM3 boards and Springware boards.

See the *Global Call API Programming Guide* for more information about the Global Call architecture.

Figure 1. Global Call Architecture When Using E1/T1 CAS/R2





## ***E1/T1 CAS/R2 Call Scenarios***

---

**3**

The E1/T1 CAS/R2 technology supports all of the call scenarios that are described in the *Global Call API Programming Guide*. Please refer to that publication for information on call scenarios.





# E1/T1 CAS/R2-Specific Operations

---

## 4

This chapter offers advice and suggestions for programmers designing and coding Global Call E1 CAS or T1 robbed bit applications in a Linux or Windows environment. Topics include the following:

- [Call Progress and Call Analysis](#) ..... 25
- [CAS Pattern Signal Declarations](#)..... 37
- [Dynamic Trunk Configuration](#)..... 40
- [Resource Association](#) ..... 44
- [Resource Allocation and Routing](#) ..... 45
- [Alarm Handling](#)..... 49
- [Run Time Configuration of the PDKRT Call Control Library](#) ..... 56
- [Run Time Configuration of PDK Protocol Parameters](#) ..... 57
- [Determining the Protocol Version](#) ..... 59

## 4.1 Call Progress and Call Analysis

Call analysis consists of both pre-connect and post-connect information about the progress of the call. Pre-connect call progress determines the status of the call connection, that is, busy, no dial tone, no ringback, etc. Post-connect call analysis, which is also known as media type detection, determines the destination party's media type, that is, answering machine, fax, voice, etc. The term *call progress analysis* (CPA) is used to refer to call progress and call analysis collectively.

**Note:** For CAS (not R2) protocols, separate pre-connect CPA needs to be carried out (initiated) since the CAS signaling itself does not indicate the condition of the line at any point of call establishment. For R2 protocols, the signaling is more intelligent. When using R2 compelled signaling, the Group B tones provide information on the condition of the line as discussed in [Section 1.5.2, “R2MF Signal Meanings”](#), on page 18.

For R2 protocols, if digits are being sent using R2 tones, pre-connect CPA is part of R2 signaling. Pre-connect CPA such as NoAnswer can only be initiated after the R2 signaling has been issued. However, R2 protocols can be configured to dial digits using DTMF/MF, in which case R2 compelled signaling is **not** used. In this case, full pre-connect CPA is supported and must be explicitly enabled.

The following sections discuss:

- [Call Analysis with DM3 Boards](#)
- [Call Analysis with Springware Boards](#)

- [Call Analysis Functionality for PDK Protocols](#)
- [Tone Definitions for PDK Protocols](#)
- [Call Analysis Functionality for ICAPI Protocols](#)

### 4.1.1 Call Analysis with DM3 Boards

**Note:** When using DM3 boards, Global Call provides a consistent method of pre-connect call progress and post-connect call analysis across analog, CAS, and ISDN protocols. Refer to the *Global Call API Programming Guide* for information about this method of CPA.

The information included below is specific to the E1/T1 CAS technology and is provided for backward compatibility only. For new applications, it is recommended to use the cross-technology CPA method described in the *Global Call API Programming Guide*.

There are two methods available for CPA when using DM3 boards: the Global Call method and the **dx\_dial()** method.

The Global Call media detection method is especially useful for performing post-connect call analysis. When activated by setting the **GCPR\_MEDIADTECT** parameter to **GCPV\_ENABLE** for a particular channel, post-connect call analysis is performed as part of the **gc\_MakeCall()** function's operation. The **gc\_MakeCall()** function is used to place a call; the signal detector analyzes the incoming signals to perform CPA.

After the normal **gc\_MakeCall()** processing finishes and a **GCEV\_CONNECTED** event is sent, call analysis runs and generates a **GCEV\_MEDIADTECTED** event that tells the application the result of the analysis (for example, FAX, PVD, or PAMD is detected). The order in which **GCEV\_CONNECTED** and **GCEV\_MEDIADTECTED** events are received may vary; refer to the specific protocol in the *Global Call Country Dependent Parameters (CDP) for PDK Protocols Configuration Guide* for more details.

The outcome of the analysis determines the events generated and the action that can be taken as follows:

- If the call is successful, **gc\_MakeCall()** finishes and a **GCEV\_CONNECTED** event is sent, call analysis runs, and generates a **GCEV\_MEDIADTECTED** event. The **gc\_ResultValue()** and **gc\_GetCallInfo()** functions can then be used to get more information about the type of media detected, such as voice, answering machine, and fax.
- If the call is not successful—for example, there is no ringback—a **GCEV\_DISCONNECTED** event is generated and the **gc\_ResultValue()** function can be used to retrieve the reason for the failure. See the *Global Call API Library Reference* for error codes and the *gcerr.h* file for more information.

**Note:** The information above applies when using **gc\_MakeCall()** in asynchronous or synchronous mode. However, in synchronous mode, since the **gc\_MakeCall()** function must complete, the **GCEV\_MEDIADTECTED** event is generated after the call is connected.

**GCPR\_MEDIADTECT** and **GCPR\_CALLPROGRESS** parameter settings for **gc\_SetParm()** actually allow the application to specify whether pre- or post-connect call analysis or both should be activated. This method for achieving this is shown in Table 2.

**Table 2. Global Call Call Progress Settings**

	<b>GCPR_CALLPROGRESS= GCPV_DISABLE</b>	<b>GCPR_CALLPROGRESS= GCPV_ENABLE (default)</b>
<b>GCPR_MEDIADETECT= GCPV_DISABLE (default)</b>	No call progress	Pre-connect call progress <b>only</b>
<b>GCPR_MEDIADETECT= GCPV_ENABLE</b>	No call progress	Full call progress

As shown in this table, the default behavior (**GCPR\_MEDIADETECT = GCPV\_DISABLE**) disables media detection but actually activates pre-connect call progress for CAS protocols. To enable full CPA, set the **GCPR\_MEDIADETECT** parameter to **GCPV\_ENABLE** for the respective channel.

**Note:** For this Global Call media detection to work, a voice device must be attached to the line device and properly routed. Failure to do so will cause subsequent outgoing call attempts to fail.

The **GCPR\_CALLPROGRESS** parameter can be used to enable or disable pre-connect call progress. When combined with **GCPR\_MEDIADETECT**, this allows the application to specify whether to use pre-connect call progress only or full call progress. If **GCPR\_CALLPROGRESS = GCPV\_DISABLE**, there will be no call progress at all, regardless of the setting of **GCPR\_MEDIADETECT**.

Table 3 explains call analysis support via the Global Call interface. The table applies to DM3 CAS protocols with flexible routing clusters, provided that a voice device is attached to the network device. Check on a protocol-by-protocol basis, as some might not support call analysis at all.

**Table 3. Call Analysis Support on DM3 Boards with CAS**

<b>Call Analysis Feature</b>	<b>Support on DM3</b>	<b>How Obtained/Notes</b>
Busy	Yes	Upon DISCONNECT event, call <b>gc_ResultValue( )</b> .
No ringback	No	
SIT	Yes	Upon DISCONNECT event, call <b>gc_ResultValue( )</b> .
No answer	Yes	Upon DISCONNECT event, call <b>gc_ResultValue( )</b> .
Cadence break	No	
Discarded	No	
NA	Yes	Use <b>GCPR_MEDIADETECT</b> parameter. Upon MEDIADETECTED event, call <b>gc_GetCallInfo( )</b> .
Unknown	Yes	Use <b>GCPR_MEDIADETECT</b> parameter. Upon MEDIADETECTED event, call <b>gc_GetCallInfo( )</b> .
PVD	Yes	Use <b>GCPR_MEDIADETECT</b> parameter. Upon MEDIADETECTED event, call <b>gc_GetCallInfo( )</b> .
PAMD	Yes	Use <b>GCPR_MEDIADETECT</b> parameter. Upon MEDIADETECTED event, call <b>gc_GetCallInfo( )</b> .

Table 3. Call Analysis Support on DM3 Boards with CAS (Continued)

Call Analysis Feature	Support on DM3	How Obtained/Notes
Fax	Yes	Use <b>GCPR_MEDIADETECT</b> parameter. Upon MEDIADETECTED event, call <b>gc_GetCallInfo()</b> .
In progress	Yes	Use <b>GCPR_MEDIADETECT</b> parameter. Upon MEDIADETECTED event, call <b>gc_GetCallInfo()</b> .

Note that the call analysis time-out parameters values apply, and they are configurable by the user. (They cannot be changed at runtime.) The parameters are **CaSignalTimeout**, **CaAnswerTimeout**, and **CaPvdTimeout**; their values are found in the CHP section of the configuration (.config) file. However, they apply only to post-connect call analysis and are not used until the call moves from an initiated to a Proceeding, Alerting, or Connected state.

Another option for call analysis is provided by the Voice API, which provides post-connect call analysis on DM3 boards through the **dx\_dial()** function. Note that the Global Call method and the **dx\_dial()** method are mutually exclusive, so you must choose one or the other.

### 4.1.2 Call Analysis with Springware Boards

The **gc\_GetCallInfo()** function is used immediately following the receipt of a GCEV\_CONNECTED event to retrieve this post-connect information notifying of the media type of the answering party. See the *Global Call API Library Reference* for more information.

Call analysis tones such as dial tone, ringback, busy, and fax are defined either in the firmware (Global Tone Detection and Global Tone Generation), or in the country dependent parameters (.cdp) file, or a combination of both. Tones defined in the firmware can be enabled or disabled by configuring parameters in the DX\_CAP (call analysis parameter) data structure. Similarly, the DX\_CAP data structure can be used to configure the voice detection algorithm that distinguishes answering machine or human speech. The default parameter values defined in the DX\_CAP data structure can be changed by the **gc\_LoadDxParm()** function to fit the needs of your application. For a detailed description of enhanced call analysis (PerfectCall) and how to use call analysis, see the *Voice API Programming Guide*. For a detailed description of the **gc\_LoadDxParm()** function, see the *Global Call API Library Reference*.

Some example uses of call progress tones are as follows:

- By detecting the ringback tone, the Global Call API can count the rings and report a GCEV\_DISCONNECTED event when the call is not answered within the specified number of rings.
- For telephone circuits that include analog links, the local line may not have access to all of the digital signaling information. If so, the user must modify the .cdp file accordingly to detect or generate the busy, ringback, or dial tone of the native country.

## Global Tone Detection (GTD) Tone Considerations

Global Call will delete all tones and load internally required tones (used for call progress) under either of the following circumstances:

- If there is a voice device attached to the network device during **gc\_OpenEx()**
- When **gc\_Attach()** or **gc\_AttachResource()** is called, if at least one of the following statements is true:
  - A voice resource is being attached to a network device opened in the PDK library (either implicitly via **gc\_OpenEx()**, or explicitly via **gc\_Attach()** or **gc\_AttachResource()**).
  - Downloading of tones is enabled (**gc\_SetParm(ldev, GCPR\_LOADTONES, GCPV\_ENABLE)**).

If Global Call deleted all tones during **gc\_OpenEx()**, **gc\_Attach()**, or **gc\_AttachResource()** as described above, then the application must reload any tones that it has loaded. It is recommended that the application not download tones for a voice device prior to calling **gc\_OpenEx()** if the voice device is specified in the **gc\_OpenEx()**, as the tones will be deleted. Similar considerations apply to **gc\_Attach()** and **gc\_AttachResource()**.

It is the application's responsibility to ensure that the internally required tones are available to the protocol during call setup. This can be done by either:

- Never deleting all tones, or
- If the application has deleted all tones while the voice resource is not attached, enabling downloading of tones

**Caution:** The application must not delete all tones while the voice resource is attached.

In any case, the application may not delete internally required tones during call setup.

**Note:** For PDK and ICAPI protocols, the tone IDs for any additional tones that must be redefined after calling **gc\_Attach()** or **gc\_AttachResource()** cannot be in the range from 101 to 189.

The overhead of downloading tones is expensive. Therefore, for any application that calls **gc\_Attach()** or **gc\_AttachResource()** several times on the same device (for example, when resource sharing), this overhead can be avoided by calling **gc\_SetParm(ldev, GCPR\_LOADTONES, GCPV\_DISABLE)**. This **gc\_SetParm()** function should be called after the call to the **gc\_Attach()** or **gc\_AttachResource()** function, or after the call to the **gc\_OpenEx()** function if the voice device is specified in **gc\_OpenEx()**. It is then the application's responsibility not to delete all tones on the voice device.

### 4.1.3 Call Analysis Functionality for PDK Protocols

Call analysis functionality for PDK protocols is discussed in the following sections:

- [Overview of Call Analysis When Using PDK Protocols](#)
- [Configuring Call Analysis in the Protocol .cdp File](#)
- [Enabling the GCEV\\_MEDIADETECTED Event](#)
- [Retrieving the Detected Media Type](#)

### 4.1.3.1 Overview of Call Analysis When Using PDK Protocols

On both DM3 and Springware boards, when using PDK protocols, media detection (i.e., call analysis) is completed in two parts: protocol and library.

There are parameters in the .cdp file that provide options for media detection with the protocol. The parameters are **PSL\_CAMediaDetectOverride** (for DM3 boards) and **PSL\_MakeCall\_MediaDetect** (for Springware boards). Further information about these parameters is given in [Section 4.1.3.2, “Configuring Call Analysis in the Protocol .cdp File”](#), on page 30.

After the protocol sends the call analysis result to the library, the library determines whether to send GCEV\_MEDIADETECTED to the application, independent of these PSL\_ parameter settings. The GCEV\_MEDIADETECTED event is disabled by default in the library, so the application must explicitly enable the event. See [Section 4.1.3.3, “Enabling the GCEV\\_MEDIADETECTED Event”](#), on page 31.

After receiving GCEV\_MEDIADETECTED (or after receiving GCEV\_CONNECTED if GCEV\_MEDIADETECTED is not enabled), the **gc\_GetCallInfo()** function is used to retrieve information about the detected media type. See [Section 4.1.3.4, “Retrieving the Detected Media Type”](#), on page 32.

### 4.1.3.2 Configuring Call Analysis in the Protocol .cdp File

PDK protocols configure default call analysis operation through the use of two Protocol Service Layer (PSL) parameters in the protocol .cdp file (the parameter names are different for DM3 and Springware boards):

- **PSL\_CACallProgressOverride** (parameter for DM3)  
**PSL\_MakeCall\_CallProgress** (parameter for Springware): Provides default options for call progress. Possible values are:
  - 0 (Always Off): Specifies that the call progress resource cannot be used by the protocol. This is the default value if this parameter is left undefined in the .cdp file.
  - 1 (Preferred): Specifies that the call progress resource is preferred by the protocol. This value is typically used for T1 and analog protocols. However, the protocol is able to function without call progress.
  - 2 (Pass-through): Specifies that the call progress resource is configured as specified dynamically by the application, for example, via **gc\_MakeCall()** when using Global Call. This value is typically used by E1 protocols.
- **PSL\_CAMediaDetectOverride** (parameter for DM3)  
**PSL\_MakeCall\_MediaDetect** (parameter for Springware): Provides options for media detection. Possible values are:
  - 1 (Preferred): Specifies that the media detection resource is preferred by the protocol. This setting is typically used for T1 and analog protocols. The protocol is able to function without media detection.
  - 2 (Pass-through): Specifies that the media detection resource is configured as specified dynamically by the application, for example, via **gc\_MakeCall()** or **gc\_SetParm()** when using Global Call. This value is typically used by E1 protocols. This is the default value if this parameter is left undefined in the .cdp file.

When call progress or media detection support PSL parameters are specified as pass-through values in the .cdp file, the application is permitted to define call analysis settings, for example via **gc\_MakeCall()** when using Global Call. More specifically:

- When the **PSL\_CACallProgressOverride** (DM3) or **PSL\_MakeCall\_CallProgress** (Springware) parameter in the .cdp file is specified as 2 (Pass-through), the application may disable call progress (the default is enabled) in its call to **gc\_MakeCall()**.
- When the **PSL\_CAMediaDetectOverride** (DM3) or **PSL\_MakeCall\_MediaDetect** (Springware) parameter in the .cdp file is **not** specified as 1 (Preferred, the default is 2, Pass-through), the application may enable media type detection (the default is disabled) in a call to **gc\_MakeCall()** or **gc\_SetParm()**.
- When call progress or media detection support PSL parameters are specified as pass-through values in the .cdp file, the application defines call analysis and/or media detection on a per call basis via the **gc\_MakeCall()** or **gc\_SetParm()** call.
- When call analysis behavior is not specified via PSL parameters in the .cdp file, the default behavior has call progress always disabled and media type detection disabled by default unless the application explicitly enables media type detection via the **gc\_MakeCall()** or **gc\_SetParm()**.
- If the call progress and/or media type detection parameters are specified in the .cdp file as 1 (Preferred) or 0 (Always Off), application setting requests (for example, the settings specified via **gc\_MakeCall()** or **gc\_SetParm()**) are ignored.

#### 4.1.3.3 Enabling the GCEV\_MEDIADETECTED Event

On DM3 and Springware boards, PDK protocols support a method of call progress configuration using the **gc\_SetConfigData()** / **gc\_SetParm()** function. The parameter used to specify call analysis (media detection) in this case is **GCPR\_MEDIADETECT**. (See [Table 2, “Global Call Call Progress Settings”](#), on page 27.) This enables media type detection on a **per channel** basis.

When this method is used to enable media type detection, a GCEV\_MEDIADETECTED event is returned to the application on media type detection so that the **gc\_GetCallInfo()** function can be used immediately to get information about the type of connection. The application does not have to wait for a GCEV\_CONNECTED event.

Note that if this method of call progress configuration is **not** used and only **PSL\_CAMediaDetectOverride** / **PSL\_MakeCall\_MediaDetect** is enabled for media detection, the application must receive a GCEV\_CONNECTED event **before** the **gc\_GetCallInfo()** function can be used to get information about the type of connection. Even after the GCEV\_CONNECTED event is received, the call information may not be available. (In this situation, **gc\_GetCallInfo()** returns GCCT\_NA.) Consequently, the application may need to poll for the information.

On Springware boards, PDK protocols support another method of call analysis via the **gc\_MakeCall()** function. The **gc\_MakeCall()** function uses the **flags** parameter in the PDK\_MAKECALL\_BLK structure to determine if call progress and/or media type detection are enabled on a **per call** basis. The two flags are NO\_CALL\_PROGRESS and MEDIA\_TYPE\_DETECT. The default values are such that call progress is enabled and media type detection is disabled, but the bits in the **flags** parameter can be changed to enable/disable call progress and/or media type detection as required. (See [PDK\\_MAKECALL\\_BLK](#) in [Chapter 9, “E1/T1 CAS/R2-Specific Data Structures”](#).) If this method is used for media detection, the



application must receive a GCEV\_CONNECTED event **before** the `gc_GetCallInfo()` function can be used as described above.

#### 4.1.3.4 Retrieving the Detected Media Type

When the `gc_GetCallInfo()` function is used to retrieve information about the detected media type, the `info_id` parameter to the `gc_GetCallInfo()` function must be `CONNECT_TYPE`. The values that may be returned when the `info_id` parameter is `CONNECT_TYPE` include:

- GCCT\_CAD  
Connection due to cadence break
- GCCT\_PVD  
Connection due to voice detection
- GCCT\_PAMD  
Connection due to answering machine detection
- GCCT\_FAX  
Connection due to fax machine detection
- GCCT\_NA  
Connection type is not available

Whether a positive media detection result is sufficient to signal a call state change to the `CONNECTED` state is dependent upon the specific PDK protocol. For example, in PDK protocols where CAS signaling is required for identifying a connection, a signaling bit change must be received before signaling a `CONNECTED` call state change. For increased flexibility, a separate .cdp file parameter, **CDP\_Connect\_Upon\_Media**, may be defined in the associated parameter file and used inside the protocol to enable the protocol to perform a call state change to the `Connected` state immediately upon positive media detection. This parameter is mostly of interest to T1 protocols.

#### 4.1.4 Tone Definitions for PDK Protocols

On Springware boards, call analysis and progress tones are mapped to US specified tones by default. PDK protocols also permit call analysis and progress tones to be customized for non-US defaults via **PSL\_TONE\_CP\_xxx** (where xxx is the call analysis tone type, that is, `BUSY`, `RINGBACK`, etc.) parameters as specified in the protocol .cdp file.

The format of a tone definition in the .cdp file is as follows:

```
ALL TONE_t TONE_<NAME> = Frequency_1, Frequency_1_Deviation, Frequency_2, Frequency_2_Deviation,
Amplitude_1, Amplitude_2, OnTime, OnTime_Deviation, OffTime, OffTime_Deviation, Mode, Repeat
Count
```

There are two basic types of tone detection for both single and dual tones: edge detection and cadence detection.

Tone detection using the edge detection algorithm provides notification either when the energy in the specified frequency band(s) exceeds the threshold (leading-edge detection) or no longer



exceeds the threshold (trailing-edge detection). Edge detection is identified by assigning a value of zero (0) to the **On Time** parameter. See Table 4 below.

Tone detection using the cadence detection algorithm provides notification when the energy in the specified frequency band(s) exceeds threshold and meets the timing requirements of the specified ring cadence. Cadence detection, like edge detection, can provide notification either when the cadence completes the specified number of cycles (**Repeat Count** parameter) or when the cadence ceases after ringing the specified number of cycles. Cadence detection is identified by assigning a non-zero value to the **On Time** parameter.

Another tone detection feature is the ability to debounce the leading edge of the tone. Rather than notifying the protocol immediately when the leading edge of the tone is detected, the protocol can specify to wait for a period of time (debounce time) before the tone signal is delivered to the protocol, that is, debouncing. This type of tone detection can be specified in the tone template as:

- **On Time:** plus half the debounce time
- **On Time Deviation:** minus half the debounce time
- **Off Time:** 0
- **Off Time Deviation:** 0
- **Repeat Count:** 0

**Note:** Many Springware boards cannot detect dual tones with frequency components closer than 65 Hz. In these instances, use a single tone template with the specified frequency band (that is, Frequency 1 +/- Frequency 1 Deviation) encompassing both dual tone ranges.

The meaning of each argument of a tone definition is explained in Table 4.

**Table 4. TONE\_t Signal Definition Parameters**

Parameter Number	Name	Description	Detect/Generate	Edge/Cadence Detection
1	Frequency 1	Frequency of first tone (in Hz)	Detect, Generate	Edge, Cadence
2	Frequency 1 Deviation	Frequency deviation for first tone (in Hz) <b>Note:</b> The minimum recommended value for this parameter is 50.	Detect	Edge, Cadence
3	Frequency 2	Frequency of second tone (in Hz)	Detect, Generate	Edge, Cadence
4	Frequency 2 Deviation	Frequency deviation for second tone (in Hz) <b>Note:</b> The minimum recommended value for this parameter is 50.	Detect	Edge, Cadence
5	Amplitude 1	Amplitude of first tone (in dB)	Generate	Neither
6	Amplitude 2	Amplitude of second tone (in dB)	Generate	Neither
7	On Time	On duration (in milliseconds) <b>Note:</b> The minimum recommended value is 50.	Detect, Generate	Cadence

Table 4. TONE\_t Signal Definition Parameters (Continued)

Parameter Number	Name	Description	Detect/Generate	Edge/Cadence Detection
8	On Time Deviation	On time deviation (in milliseconds) <b>Note:</b> The minimum recommended value is 50.	Detect	Cadence
9	Off Time	Off duration (in milliseconds) <b>Note:</b> The minimum recommended value is 50.	Detect, Generate	Cadence
10	Off Time Deviation	Off time deviation (in milliseconds) <b>Note:</b> The minimum recommended value is 50.	Detect	Cadence
11	Mode	<p>Detection notification. For <b>Springware</b>, the values and their meanings are:</p> <ul style="list-style-type: none"> <li>0 – Report a tone match at the termination of the tone. For edge detection, this is the trailing edge. For cadence detection, this is the termination of the cadence after the specified number of cycles.</li> <li>1 – Report a tone match at the beginning of the tone. For edge detection, this is the leading edge. For cadence detection, this is the onset of cadence detection.</li> </ul> <p>For <b>DM3</b>, the values and their meanings are:</p> <ul style="list-style-type: none"> <li>0 – Report a tone match at the termination of the tone. For edge detection, this is the trailing edge. For cadence detection, this is after cadence has ended.</li> <li>1 – Report a tone match at the beginning of the tone. For edge detection, this is the leading edge. For cadence detection, this is after the first pulse has been detected and at the beginning of the second pulse. (Normal cadence detection in DM3 requires at least 2 pulses.)</li> <li>2 – Report a tone match at both the beginning and end.</li> <li>3 – Disable tone detection. This is used for definitions intended only for tone generation.</li> <li>4 – Report a tone match at the beginning of the tone. For edge detection, this is the leading edge. For cadence detection, this is after the first pulse has been detected. (Unlike mode 1, this mode does not require 2 pulses.)</li> </ul>	Detect	Edge, Cadence
12	Repeat Count	Repetition count (the number of repetitions on cycles)	Detect, Generate	Cadence

If TONE\_x is previously defined, TONE\_y may be set equal to TONE\_x in the following manner:

```
ALL TONE_t TONE_y = TONE_x
```

The following are examples of tone declarations in a .cdp file:

```
/*
This defines the ringback tone. The currently defined tone is a
tone (440Hz+480Hz) on for 0.25 secs and off for 0.25 secs and a
ring count of 1
*/
R4 TONE_t TONE_RINGBACK = 440,0,480,0,0,0,250,0,250,0,0,1

/*
This identifies the KP tone for ANI.
*/
R4 TONE_t TONE_ANI_KP = 1100,0,1700,0,0,0,100,0,0,0,0,1
```

## 4.1.5 Call Analysis Functionality for ICAPI Protocols

**Note:** The information in this section is applicable to Springware boards only. DM3 boards do not use ICAPI protocols.

Global Call call analysis uses global tone detection (GTD) and timers. Some of the country dependent parameters (.cdp) files define tone templates for recognition of call progress tones. The tone IDs defined match the protocol parameter numbers (for example, parameter \$103 creates tone ID# 103). See the *Voice API Programming Guide* for information about working with and building tone templates.

Parameter \$1, \$6, or \$13 in the .cdp file defines the maximum time (in seconds) for a call to be answered. Within that interval, a busy tone and ringback tone can be detected. If the timer expires, the GCEV\_DISCONNECTED event is reported to the application.

Two separate busy tones can be defined to accommodate two different call progress failure tones (that is, busy and out-of-order). Busy tones are defined in parameters \$103 and \$104 using the following format:

```
$103: - <frequency 1> <deviation> <frequency 2> <deviation>
%01: - <on time> <on deviation> <off time> <off deviation>
%02: - <number of cycles before detect>
```

Frequency is expressed in Hz; time duration is expressed in 10 ms units; unspecified values are set to 0. The deviation value for frequency 1 or 2 specifies the allowable variation in Hz. The %01 parameter relates to cadence detection. Cadence detection analyzes the audio signal on the line to detect a repeating pattern of sound (on time) and silence (off time). The deviation value for cadence detection is the allowable variation in 10 ms units. The %02 parameter specifies the number of times that the cadence on/off pattern must be detected before classifying the tone detected.

To comment out a tone template, insert a “;” (semicolon) as the first character in all three lines of the definition. If either of the busy tones is detected, the GCEV\_DISCONNECTED event is reported to the application.

A ringback tone is defined in parameter \$105 using the format defined above. The maximum allowable time between successive rings is defined in parameter \$3 in 10 ms units. ICAPI starts a

timer after receiving a ringback TONEOFF event. Typically, Connect is indicated by line signaling. However, if the network cannot indicate a Connect via line signaling, then Connect can be indicated if the next TONEON event does not arrive before the ICAP timer expires.

To disable Connect detection, set parameter \$3 to 0. Global Call will still be able to count the rings and report the GCEV\_DISCONNECTED event if the maximum number of rings is reached. The maximum number of rings is set in parameter \$1.

The ringback tone heard on any specific call depends on the specific CO that is serving the called party, not the local CO. If the ringback tone is not known, the recommendation is to remove this tone from the country dependent parameters (.cdp) file.

Only the call progress tone definitions in the .cdp file are used by the Global Call API. The R1 and R2 tone definitions are used only if you disable R2 MF support in the *icapi.cfg* file by setting the \$17 parameter to 1.

The following are examples of the definitions of busy tones \$103 and \$104 and ringback tone \$105 in the .cdp file:

```
*****
*      TID # 103  BUSY      *
*****
$103 BUSY      : 450      35
%01 cadence    : 50 10  50 10
%02 cycle      : 2

*****
*      TID # 104  SBUSY     *
*****
$104 SBUSY     :450      35
%01 cadence    : 25  5  25  5
%02 cycle      : 3

*****
*      TND # 105  RINGBACK  *
*****
$105 RINGBACK  : 450      35
%01 cadence    : 80
```

See the *Voice API Programming Guide* for information about using cadence, cadence detection, and tone definitions for determining the progress of outbound calls.

In addition, the following outbound parameters in the .cdp file may need to be modified when using these call progress tones:

- Number of ringback tones before returning GCEV\_CALLSTATUS event with a GCRV\_NOANSWER result value (typically, parameter \$1 or \$5)
- Default maximum time in seconds for a call to be answered (typically, parameter \$1, \$6, or \$13)

After the .cdp file is modified as described above, whenever one of the defined conditions is detected on a channel, the **gc\_MakeCall()** function is terminated with a busy, no answer, or time-out result/error value.

For some ICAPI protocols, certain parameters must be set in the .cdp file to ensure proper operation of the protocol. Refer to the *Global Call Country Dependent Parameters (CDP) for ICAPI Protocols Reference* for country dependent parameters that are most likely to be modified and for any required settings.

**Note:** For ICAPI protocols, the filename specified after @0 in the .cdp file must also be specified in the *country.c* file used in Linux applications.

## 4.2 CAS Pattern Signal Declarations

CAS signals are defined in the cdp file for each protocol. They are typically set to default values based on protocol specifications, but can be tuned if needed. The following sections describe the formats of CAS signal declarations for different types of signals:

- CAS\_SIGNAL\_TRANS\_t
- CAS\_SIGNAL\_PULSE\_t
- CAS\_SIGNAL\_TRAIN\_t

- Notes:**
1. When editing CAS signals in the .cdp file, CAS signal *x* may be set equal to CAS signal *y*, for example: All CAS\_SIGNAL\_TRANS\_t CAS\_x = CAS\_y
  2. In the signal definitions:
    - When the term “minimum” is used, this implies the information is used for detection and represents a minimum time for which the associated signal must occur.
    - When the term “maximum” is used, this implies the information is used for detection and represents the maximum time that the associated signal may occur.
    - When the term “nominal” is used, this implies the information is used for generation and represents the actual time to transmit the associated signal.

### 4.2.1 CAS\_SIGNAL\_TRANS\_t

CAS\_SIGNAL\_TRANS\_t signal declarations represent a transition from one signaling pattern to another. The format of a CAS\_SIGNAL\_TRANS\_t signal definition in the .cdp file is:

```
CAS_SIGNAL_TRANS_t format = PreTrans, PostTrans, PreTransInterval, PostTransInterval,
PreTransIntervalNominal, PostTransIntervalNominal
```

Some examples are:

```
R4 CAS_SIGNAL_TRANS_t CAS_BLOCK = 00xx,11xx,50,50,80,80
R4 CAS_SIGNAL_TRANS_t CAS_UNBLOCK = 11xx,00xx,50,50,80,80
ALL CA_ SIGNAL_TRANS_t CAS_SEIZE = xxx, 10xx
```

The meaning of each argument is explained in Table 5.

Table 5. CAS\_SIGNAL\_TRANS\_t Signal Definition Parameters

Parameter Number	Name	Description
1	PreTrans	$B_a B_b B_c B_d$ where $B_i$ represents a signaling bit (0, 1, - = don't care, or x = don't care) for bit i, where i = a, b, c, or d. <b>Note:</b> Although T1 signaling does not have c and d bits, they must be specified for T1 protocols as don't care values.
2	PostTrans	$B_a B_b B_c B_d$ (See description of PreTrans.)
3	PreTransInterval	Minimum time for the duration of the pre-transition interval. If the value is -1 or not present, then the global timing parameter PSL_PreTransInterval is used.
4	PostTransInterval	Minimum time for the duration of the post-transition interval. 0 is allowed. If the value is -1 or is not present, then the global timing parameter PSL_PostTransInterval is used.
5	PreTransIntervalNominal	Nominal time for the duration of the pre-transition interval. If the value is -1 or is not present, then the global timing parameter PSL_PreTransIntervalNominal is used. <b>Note:</b> For DM3 boards, this value is always ignored, and the PreTransInterval value is used.
6	PostTransIntervalNominal	Nominal time for the duration of the post-transition interval. If the value is -1 or is not present, then the global timing parameter PSL_PostTransIntervalNominal is used. <b>Note:</b> For DM3 boards, this value is always ignored, and the PostTransInterval value is used.
<b>Notes:</b> Time intervals are specified in units of 1 millisecond. The actual granularity is implementation dependent, as is the maximum value. Due to implementation restrictions, no time value should be less than 20 milliseconds (except where 0 is allowed).		

## 4.2.2 CAS\_SIGNAL\_PULSE\_t

CAS\_SIGNAL\_PULSE\_t signal declarations represent a transition from one signaling pattern to another and then back to the original signaling pattern. The format of a CAS\_SIGNAL\_PULSE\_t signal definition in the .cdp file is:

```
CAS_SIGNAL_PULSE_t format = OffPulse, OnPulse, PrePulseInterval, PostPulseInterval,
PrePulseIntervalNominal, PostPulseIntervalNominal, PulseIntervalMin, PulseIntervalNominal,
PulseIntervalMax
```

Some examples are:

```
R4 CAS_SIGNAL_PULSE_t CAS_WINK = 00xx,11xx,50,50,80,80,200,250,300
R4 CAS_SIGNAL_PULSE_t CAS_SEIZE_ACK = 00xx,11xx,50,50,80,80,200,250,300
```

The meaning of each argument is explained in Table 6.

Table 6. CAS\_SIGNAL\_PULSE\_t Signal Definition Parameters

Parameter Number	Name	Description
1	OffPulse	$B_a B_b B_c B_d$ where $B_i$ represents a signaling bit (0, 1, - = don't care, or x = don't care) for bit i, where i = a, b, c, or d. <b>Note:</b> Although T1 signaling does not have c and d bits, they must be specified for T1 protocols as don't care values.
2	OnPulse	$B_a B_b B_c B_d$ (See description of OffPulse.)
3	PrePulseInterval	Minimum time for the duration of the pre-pulse interval. 0 is allowed.
4	PostPulseInterval	Minimum time for the duration of the post-pulse interval. 0 is allowed.
5	PrePulseIntervalNominal	Nominal time for the duration of the pre-pulse. 0 is allowed. <b>Note:</b> For DM3 boards, this value is always ignored, and the PrePulseInterval value is used.
6	PostPulseIntervalNominal	Nominal time for the duration of the post-pulse. 0 is allowed. <b>Note:</b> For DM3 boards, this value is always ignored, and the PostPulseInterval value is used.
7	PulseIntervalMin	Minimum time for the duration of the pulse.
8	PulseIntervalNominal	Nominal time for the duration of the pulse.
9	PulseIntervalMax	Maximum time for the duration of the pulse.
<b>Notes:</b> Time intervals are specified in units of 1 millisecond. The actual granularity is implementation dependent, as is the maximum value. Due to implementation restrictions, no time value should be less than 20 milliseconds (except where 0 is allowed).		

### 4.2.3 CAS\_SIGNAL\_TRAIN\_t

CAS\_SIGNAL\_TRAIN\_t signal declarations represent a “train,” that is, a sequence of pulses. The format of a CAS\_SIGNAL\_TRAIN\_t signal definition in the .cdp file is:

```
CAS_SIGNAL_TRAIN_t format = OffPulse, OnPulse, PreTrainInterval, PostTrainInterval,
PreTrainIntervalNominal, PostTrainIntervalNominal, PulseIntervalMin, PulseIntervalNominal,
PulseIntervalMax, InterPulseIntervalMin, InterPulseIntervalNominal, InterPulseIntervalMax
```

The meaning of each argument is explained in Table 7.

Table 7. CAS\_SIGNAL\_TRAIN\_t Signal Definition Parameters

Parameter Number	Name	Description
1	OffPulse	$B_a B_b B_c B_d$ where $B_i$ represents a signaling bit (0, 1, - = don't care, or x = don't care) for bit i, where i = a, b, c, or d. <b>Note:</b> Although T1 signaling does not have c and d bits, they must be specified for T1 protocols as don't care values.
2	OnPulse	$B_a B_b B_c B_d$ (See description of OffPulse.)
3	PreTrainInterval	Minimum time for the duration of the pre-train interval. 0 is allowed.
4	PostTrainInterval	Minimum time for the duration of the post-train interval. Must be greater than or equal to InterPulseIntervalMax + 20.
5	PreTrainIntervalNominal	Nominal time for the duration of the pre-train interval. <b>Note:</b> For DM3 boards, this value is always ignored, and the PreTrainInterval value is used.
6	PostTrainIntervalNominal	Nominal time for the duration of the post-train interval. <b>Note:</b> For DM3 boards, this value is always ignored, and the PostTrainInterval value is used.
7	PulseIntervalMin	Minimum time for the duration of the pulse.
8	PulseIntervalNominal	Nominal time for the duration of the pulse.
9	PulseIntervalMax	Maximum time for the duration of the pulse.
10	InterPulseIntervalMin	Minimum time for the duration of the interval between pulses.
11	InterPulseIntervalNominal	Nominal time for the duration of the interval between pulses.
12	InterPulseIntervalMax	Maximum time for the duration of the interval between pulses.
<b>Notes:</b> Time intervals are specified in units of 1 millisecond. The actual granularity is implementation dependent, as is the maximum value. Due to implementation restrictions, no time value should be less than 20 milliseconds (except where 0 is allowed).		

## 4.3 Dynamic Trunk Configuration

When using DM3 boards, Global Call provides the ability to perform the following dynamic configuration operations at run time:

- [Setting the Line Type and Coding for a Trunk](#)
- [Specifying the Protocol for a Trunk](#)

**Note:** The `gc_SetConfigData()` function can be used on a board device to perform these operations. However, it is the application's responsibility to handle all active calls on the trunk, and terminate them if necessary. In addition, the `gc_ResetLineDev()` function may be issued on all channels (time slots) prior to issuing `gc_SetConfigData()` to prevent incoming calls. If there are any active



calls present at the time the **gc\_ResetLineDev()** or **gc\_SetConfigData()** function is issued, they are gracefully terminated internally. The application does not receive GCEV\_DISCONNECTED events when calls are terminated in this manner.

### 4.3.1 Setting the Line Type and Coding for a Trunk

**Note:** This feature is only applicable when using DM3 boards.

The **gc\_SetConfigData()** function uses a GC\_PARM\_BLK structure that contains the configuration information. The GC\_PARM\_BLK is populated using the **gc\_util\_insert\_parm\_val()** function.

To configure the line type, use the **gc\_util\_insert\_parm\_val()** function with the following parameter values:

- **parm\_blkpp** = pointer to the address of a valid GC\_PARM\_BLK structure where the parameter and value are to be inserted
- **setID** = CCSET\_LINE\_CONFIG
- **parmID** = CCPARM\_LINE\_TYPE
- **data\_size** = sizeof(int)
- **data** = One of the following values:
  - Enum\_LineType\_dsx1\_D4 - D4 framing type, Superframe (SF)
  - Enum\_LineType\_dsx1\_ESF - Extended Superframe (ESF)
  - Enum\_LineType\_dsx1\_E1 - E1 standard framing
  - Enum\_LineType\_dsx1\_E1\_CRC - E1 standard framing and CRC-4

To configure coding type, use the **gc\_util\_insert\_parm\_val()** function with the following parameter values:

- **parm\_blkpp** = pointer to the address of a valid GC\_PARM\_BLK structure where the parameter and value are to be inserted
- **setID** = CCSET\_LINE\_CONFIG
- **parmID** = CCPARM\_CODING\_TYPE
- **data\_size** = sizeof(int)
- **data** = One of the following values:
  - Enum\_CodingType\_AMI - Alternate Mark Inversion
  - Enum\_CodingType\_B8ZS - Modified AMI used on T1 lines
  - Enum\_CodingType\_HDB3 - High Density Bipolar of Order 3 used on E1 lines

Once the GC\_PARM\_BLK has been populated with the desired values, the **gc\_SetConfigData()** function can be issued to perform the configuration. The parameter values for the **gc\_SetConfigData()** function are as follows:

- **target\_type** = GCTGT\_CCLIB\_NETIF
- **target\_id** = the trunk line device handle, as obtained from **gc\_OpenEx()** with a **devicename** string of “:N\_dtiBx:P..”.

- **target\_datap** = GC\_PARM\_BLK\_P parameter pointer, as constructed by the utility function **gc\_util\_insert\_parm\_val()**
- **time\_out** = time interval (in seconds) during which the target object must be updated with the data. If the interval is exceeded, the update request is ignored. This parameter is supported in synchronous mode only, and it is ignored when set to 0.
- **update\_cond** = GCUPDATE\_IMMEDIATE
- **request\_idp** = pointer to the location for storing the request ID
- **mode** = EV\_ASYNC for asynchronous execution or EV\_SYNC for synchronous execution

The application receives one of the following events:

- GCEV\_SETCONFIGDATA to indicate that the request to dynamically change the line type and/or coding has been successfully initiated.
- GCEV\_SETCONFIGDATAFAIL to indicate that the request to dynamically change the line type and/or coding failed. More information is available from the GC\_RTCM\_EVTDATA structure associated with the event.

The following code example shows how to dynamically configure a T1 trunk to operate with the Extended Superframe (ESF) line type and the B8ZS coding type.

```
GC_PARM_BLK_P ParmBlkp = NULL;
long id;

/* configure Line Type = Extended Superframe for a T1 trunk */
gc_util_insert_parm_val(&ParmBlkp, CCSET_LINE_CONFIG, CCPARM_LINE_TYPE, sizeof(int),
    Enum_LineType_dsx1_ESF);

/* configure Coding Type = B8ZS for a T1 trunk */
gc_util_insert_parm_val(&ParmBlkp, CCSET_LINE_CONFIG, CCPARM_CODING_TYPE, sizeof(int),
    Enum_CodingType_B8ZS);

gc_SetConfigData(GCTGT_CCLIB_NETIF, bdev, ParmBlkp, 0, GCUPDATE_IMMEDIATE, &id, EV_ASYNC);
gc_util_delete_parm_blk(ParmBlkp);

if (sr_waitevt(-1) >= 0)
{
    METAEVENT meta;
    gc_GetMetaEvent(&meta);
    switch(sr_getevtttype())
    {
        case GCEV_SETCONFIGDATA:
            printf("Received event GCEV_SETCONFIGDATA(ReqID=%d) on device %s\n", ((GC_RTCM_EVTDATA *) (meta.evtdatap))->request_ID,
                ATDV_NAMEP(sr_getevtdev()));
            break;
        case GCEV_SETCONFIGDATA_FAIL:
            printf("Received event GCEV_SETCONFIGDATAFAIL(ReqID=%d) on device %s, Error=%s\n", ((GC_RTCM_EVTDATA *) (meta.evtdatap))->request_ID,
                ATDV_NAMEP(sr_getevtdev()),
                ((GC_RTCM_EVTDATA *) (meta.evtdatap))->additional_msg);
            break;
        default:
            printf("Received event 0x%x on device %s\n", sr_getevtttype(),
                ATDV_NAMEP(sr_getevtdev()));
            break;
    }
}
```

### 4.3.2 Specifying the Protocol for a Trunk

**Note:** This feature is only applicable when using DM3 boards.

The protocol used by a trunk can be dynamically configured by using the **gc\_SetConfigData()** function after devices have been opened. All channels on the affected trunk inherit the newly selected protocol.

The **gc\_SetConfigData()** function uses a GC\_PARM\_BLK structure that contains the configuration information. The GC\_PARM\_BLK is populated using the **gc\_util\_insert\_parm\_ref()** function.

To configure the protocol, use the **gc\_util\_insert\_parm\_ref()** function with the following parameter values:

- **parm\_blkpp** = pointer to the address of a valid GC\_PARM\_BLK structure where the parameter and value are to be inserted
- **setID** = GCSET\_PROTOCOL
- **parmID** = GCPARM\_PROTOCOL\_NAME
- **data\_size** = strlen("<protocol\_name>"), for example strlen("pdk\_ar\_r2\_io")
- **data** = "<protocol\_name>", for example, "pdk\_ar\_r2\_io" (a null-terminated string). For CAS/R2 protocols, this is the name of the CDP file (without the .cdp extension) of the protocol variant being selected. This protocol variant must already be downloaded, i.e., it must already be specified in the *pdk.cfg* file.

Once the GC\_PARM\_BLK has been populated with the desired values, the **gc\_SetConfigData()** function can be issued to perform the configuration. The parameter values for the **gc\_SetConfigData()** function are as follows:

- **target\_type** = GCTGT\_CCLIB\_NETIF
- **target\_id** = the trunk line device handle, as obtained from **gc\_OpenEx()** with a **devicename** string of ":N\_dtiBx:P..".
- **target\_datap** = GC\_PARM\_BLK parameter pointer, as constructed by the utility function **gc\_util\_insert\_parm\_ref()**
- **time\_out** = time interval (in seconds) during which the target object must be updated with the data. If the interval is exceeded, the update request is ignored. This parameter is supported in synchronous mode only, and it is ignored when set to 0.
- **update\_cond** = GCUPDATE\_IMMEDIATE
- **request\_idp** = pointer to the location for storing the request ID
- **mode** = EV\_ASYNC for asynchronous execution or EV\_SYNC for synchronous execution

The application receives one of the following events:

- GCEV\_SETCONFIGDATA to indicate that the request to dynamically change the protocol has been successfully initiated.
- GCEV\_SETCONFIGDATAFAIL to indicate that the request to change the protocol has failed. More information is available from the GC\_RTCM\_EVTDATA structure associated with the event.

The following code example shows how to dynamically configure a trunk to operate with the `pdk_ar_r2_io` protocol.

```
static int MAX_PROTOCOL_LEN=20;
GC_PARM_BLK ParmBlkp = NULL;
long id;
char protocol_name[]="pdk_ar_r2_io";

gc_util_insert_parm_ref(&ParmBlkp, GCSET_PROTOCOL, GCPARM_PROTOCOL_NAME,
strlen(protocol_name)+1, protocol_name);

gc_SetConfigData(GCTGT_CCLIB_NETIF, bdev, ParmBlkp, 0, GCUPDATE_IMMEDIATE, &id, EV_ASYNC);
gc_util_delete_parm_blk(ParmBlkp);

if (sr_waitevt(-1) >= 0)
{
    METAEVENT meta;
    gc_GetMetaEvent(&meta);

    switch(sr_getevtttype())
    {
        case GCEV_SETCONFIGDATA:
            printf("Received event GCEV_SETCONFIGDATA(ReqID=%d) on device %s\n", ((GC_RTCM_EVTDATA *) (meta.evtdatap))->request_ID,
                ATDV_NAMEP(sr_getevtdev()));
            break;
        case GCEV_SETCONFIGDATA_FAIL:
            printf("Received event GCEV_SETCONFIGDATAFAIL(ReqID=%d) on device %s, Error=%s\n", ((GC_RTCM_EVTDATA *) (meta.evtdatap))->request_ID,
                ATDV_NAMEP(sr_getevtdev()),
                ((GC_RTCM_EVTDATA *) (meta.evtdatap))->additional_msg);
            break;
        default:
            printf("Received event 0x%x on device %s\n", sr_getevtttype(),
                ATDV_NAMEP(sr_getevtdev()));
            break;
    }
}
```

## 4.4 Resource Association

In E1 CAS and T1 robbed bit protocols, a combination of line signaling and audio tones are used to establish a call. The line signaling is controlled by a network time slot device, or resource, and the tones are controlled by a voice channel (voice resource). Voice channel, voice resource, and tone resource are used interchangeably in this manual when discussing Global Call functionality.

Typically, in E1 CAS or T1 robbed bit environments, a Global Call line device consists of a network time slot resource and a voice resource. When the same voice resource is always used for a given network time slot, then this configuration is called a dedicated voice resource. The Global Call line device ID is a single ID that represents the combination of the voice and network resources that work together to establish and to tear-down calls.

In configurations with more network time slot resources than available voice (or tone) resources, the application may share these available voice resources among the time slots (resource sharing). When voice resources are shared, the Global Call line device ID represents a network time slot after issuing a `gc_OpenEx()` function. However, before issuing a `gc_MakeCall()` or a `gc_WaitCall()` function, a voice resource must be attached to the Global Call line device using the

**gc\_Attach()** function and then routed to the line device's network time slot. The **gc\_Attach()** function tells the Global Call protocol handler which voice channel will be used to establish the call. Once the call is established (answered), the application can use this voice resource for other calls by first detaching the voice resource using the **gc\_Detach()** function from the current line device and then attaching this voice resource to another line device using the **gc\_Attach()** function. The **gc\_Detach()** function must not be used to detach the voice resource until the call is in the connected state.

See [Section 4.5, “Resource Allocation and Routing”](#), on page 45 for more information.

## 4.5 Resource Allocation and Routing

E1 CAS and T1 robbed bit protocols require tone generation and detection capability and therefore require a voice or tone resource for setting up a call. Application development considerations for using dedicated voice (or tone) resources or shared voice (or tone) resources in an E1 CAS and T1 robbed bit environment are discussed in the following sections:

- [Dedicated Voice Resources](#)
- [Shared Voice Resources](#)

### 4.5.1 Dedicated Voice Resources

Applications requiring voice resources during the entire call (for example, voice-mail and announcements) must have enough voice channels to dedicate one channel to each network interface time slot. Global Call simplifies applications written to handle E1 CAS and T1 robbed bit protocols using dedicated voice resource configurations. To use Global Call functionality to set up dedicated resources, the application must pass both the network time slot and the voice channel to the **gc\_OpenEx()** function. The Global Call API uses this information to automatically:

- Open the network board device, if not previously opened (the board device is used internally by Global Call)
- Open both the voice channel and the network time slot
- Route the voice channel and network time slot together (full duplex) (CT Bus configurations only)
- Associate the voice channel and the network time slot by issuing an internal **gc\_Attach()** function

For CT Bus applications, applications using dedicated voice resources (a voice resource dedicated to a network resource) do not need to route the voice and network resources together nor issue the **gc\_Attach()** function before making a call or when handling a pending call. For applications using shared voice resources, the voice resource must be attached to a network resource before call establishment. After call establishment, this voice resource may be detached and then attached to a different network resource.

To perform activities such as routing and voice store and forward, etc., use the **gc\_GetResourceH()** function to obtain the voice and network handles associated with a line device. For example, before playing a file, you can retrieve the voice handle using the

**gc\_GetResourceH()** function. If needed, you may route other resources to the network interface (for example, to send a fax) and reroute the voice channel back to the network interface before setting up or waiting for another call. You must route the same voice channel back to the associated network interface channel because these two resources were internally attached when opened.

The following example illustrates the function calls that apply when using dedicated voice resources.

## Dedicated Voice Resources Example

```
.
.
#define MAXCHAN 30
struct linebag{
    LINEDEV  ldev;
    CRN      crn;
    INT      state;
}port[MAXCHAN+1]
.
.
/* Open a Global Call device with a voice channel and a network time slot */
1 ----> if (gc_OpenEx(&linedev, "N_dtiB1T1:P_br_r2_o:V_dxxxB1C1", 0,
    (void*)&port[port_index]) == GC_SUCCESS) {
    /*
     * Wait for GCEV_UNBLOCKED event.
     */
    .
    .
    /* Make an outgoing call */
2 ----> if (gc_MakeCall(linedev, &crn, "123456", NULL, 0,
    EV_ASYNC) == GC_SUCCESS) {
    /*
     * Wait for GCEV_CONNECTED event.
     */
    } else {
        /* Process error from gc_MakeCall( ) */
    }
} else {
    /* Process error from gc_Open( ) */
}
.
.
```

**Legend:**

- 1 The **gc\_OpenEx( )** function:
  - Opens a Global Call line device using time slot 1 of dtiB1, opens voice channel dxxxB1C1, and configures the line device to use outbound Brazilian R2 protocol
  - Opens the time slot and voice channel automatically
  - Opens the network board device automatically, if not already opened to monitor the alarm
  - Sets the user attribute, **usrattr**, (void\*)&port[port\_index] into the channel information structure
 CT Bus time slot routing and attaching are done automatically. The function need only be called once for a time slot/voice channel pair.
- 2 The **gc\_MakeCall( )** function is invoked once for each outbound call.

## 4.5.2 Shared Voice Resources

Applications requiring voice resources for a limited portion of the call, typically during call setup, may share voice resources among the available network time slots. For example, using a D/320SC voice board and two DTI/300SC (E1 interface) network boards, 32 voice channels may be able to handle the audio portions of the call control for 60 network interface time slots. This savings in hardware requires more complexity in writing the application, which must manage both the voice and network resources, and places limits on your call throughput. The number of calls that can be established simultaneously is limited to the number of voice resources in the system.

For voice resource sharing configurations, you need only specify the network time slot and protocol in the **gc\_OpenEx( )** function. This function uses the Global Call library to open the network time slot device. Your application must also open the voice device and then route and attach the necessary resources before these resources are needed for signaling. You must explicitly open the voice device by issuing a **dx\_open( )** function to open the voice device selected. For routing these devices, use the native time slot routing functions or the CT Bus **nr\_scroute( )** and **nr\_scunroute( )** functions provided for the voice and network devices. For example, route the devices using the routing functions provided by the network and voice libraries, and then use the **gc\_Attach( )** and **gc\_Detach( )** functions to associate or disassociate a voice channel and a Global Call line device and therefore a network time slot. When the above sequence of operations completes, use the **gc\_MakeCall( )** or **gc\_WaitCall( )** function, as appropriate.

After a call is answered, the voice resource can be detached from the network time slot using the **gc\_Detach( )** function and routed to another network time slot using the **nr\_scunroute( )** and **nr\_scroute( )** functions.

The following example illustrates the function calls that apply when using shared voice resources.

### Shared Voice Resources Example

```
1 ---->  if (gc_OpenEx(&linedev, ":N_dtiB1T1:P_br_r2_o", 0,
                (void*)&port[port_index]) == GC_SUCCESS) {
                /* process error */
            }
```

```

2 ---->  if (gc_GetNetworkH(linedev, &networkh) != GC_SUCCESS)
        {
            /* process error */
        }

3 ---->  if ((voiceh = dx_open("dxxxB1C1", 0)) == -1)
        {
            /* process error */
        }

4 ---->  printf("***** %d: Calling Attach %d\n", index, voiceh);
        if (gc_Attach(linedev, voiceh, EV_SYNC) != GC_SUCCESS)
        {
            /* process error */
        }

5 ---->  if (nr_scroute(networkh, SC_DTI, voiceh, SC_VOX,
        SC_FULLDUP) == -1)
        {
            /* process error */
        }
        /* Wait for GCEV_UNBLOCKED event */

6 ---->  if (gc_MakeCall(linedev, &crn, "123456", NULL, 0, EV_ASYNC)
        != GC_SUCCESS)
        {
            /* process error */
        }
        .
        .
        /*
        * Wait for GCEV_CONNECTED event. Voice resource may be detached
        * if necessary after receiving this event.
        */

7 ---->  if (gc_Detach(linedev, voiceh, EV_SYNC) != GC_SUCCESS)
        {
            /* process error */
        }

8 ---->  if (nr_scunroute(networkh, SC_DTI, voiceh, SC_VOX, SC_FULLDUP) == -1)
        {
            /* process error */
        }

```

#### Legend:

- 1    The **gc\_OpenEx( )** function:
  - opens a Global Call line device using time slot 1 of dtiB1 using outbound Brazilian R2 protocol
  - opens the network board device automatically, if not already opened
  - sets the user attribute, **usrattr**, (void\*)&port[port\_index] into the channel information structure
 The specified network time slot device is opened. This function need only be called once for a time slot.
- 2    The **gc\_GetNetworkH( )** function retrieves network device handle.



- 3 The **dx\_open()** function opens voice device and gets voice device handle.
- 4 The **gc\_Attach()** function logically connects voice and network resources.
- 5 The **nr\_scroute()** function routes voice and network resources together.
- 6 The **gc\_MakeCall()** function is invoked each time a call is to be made.
- 7 The **gc\_Detach()** function disassociates the voice resource from the Global Call line device.
- 8 The **nr\_scunroute()** function unroutes the voice and network resources.

## 4.6 Alarm Handling

Alarm handling using Global Call is different depending on the board architecture (DM3 or Springware). The following sections provide information about handling alarms in each architecture:

- [Alarm Handling for DM3 boards](#)
- [Alarm Handling for Springware Boards](#)

### 4.6.1 Alarm Handling for DM3 boards

When using DM3 boards, alarms are recognized, and can also be transmitted, on a span (trunk) basis. Once an alarm is detected, all open channels on that span receive a GCEV\_BLOCKED event. When the alarm is cleared, open channels receive a GCEV\_UNBLOCKED event.

The **gc\_SetEvtMsk()** function can be used to mask events on a line device. Using the **gc\_SetEvtMsk()** function on a line device for a time slot sets the mask for the specified time slot only and does not apply to all time slots on the same trunk as is the case when using Springware boards.

The set of Global Call functions that comprise the Global Call Alarm Management System (GCAMS) interface are supported with the following restrictions:

- Using GCAMS, the application has the ability to set which detected alarms are blocking and non-blocking as described in the *Global Call API Programming Guide*. However, this capability applies on a span basis only. Changing which alarms are blocking and non-blocking for one time slot results in changing which alarms are blocking and non-blocking for all time slots on the span.
- Using the **gc\_GetAlarmParm()** and **gc\_SetAlarmParm()** functions to retrieve and set specific alarm parameters, for example alarm triggers, is not supported.
- The **gc\_TransmitAlarms()** and **gc\_StopTransmitAlarms()** functions can be used to start and stop the transmission of alarms to the remote side. Table 8 gives the alarms that can be transmitted on E1 and T1 interfaces.

Table 8. Alarms That Can Be Transmitted on E1 and T1 Interfaces on DM3 Boards

E1 Alarm	T1 Alarm	Description	Equivalent 0x1626 Parameter Value in CONFIG Files Used for Trunk Preconditioning ‡
DEA_REMOTE †	YELLOW †	Remote alarm indication (RAI)	2
DEA_UNFRAMED1 †	BLUE †	Alarm indication signal (AIS)	1
DEA_SIGNALALL1 †	—	Signaling all 1s alarm (a multi-frame alarm)	Not applicable
DEA_DISTANTMF †	—	Distant multi-frame alarm	Not applicable
† Defines that can be used in the alarm_number field of the ALARM_FIELD structure when using the <b>gc_TransmitAlarms()</b> and <b>gc_StopTransmitAlarms()</b> functions to start and stop the transmission of specific alarms. ‡ Trunk preconditioning is the ability to place board interface trunks in an alarm state during board initialization. See the DM3 Configuration Guide for more information.			

The following list shows the detected (incoming) alarms that are supported on E1 for DM3 boards. The dagger symbol (†) next to an alarm name indicates that the alarm is blocking by default. The default can be changed using **gc\_SetAlarmConfiguration()**. For alarms where a default threshold value is shown, the default can be changed in the .config file for the board as explained in the DM3 Configuration Guide.

**DTE1\_BPVS**

Bipolar violation count saturation. The default threshold value is 255 and the range is 0 to 255.

**DTE1\_CECS**

CRC4 error count saturation. The default threshold value is 255 and the range is 0 to 255.

**DTE1\_CRC\_CFA†**

Time slot 16 CRC failure

**DTE1\_CRC\_CFAOK**

Time slot 16 CRC failure recovered

**DTE1\_ECS**

Frame sync bit error count saturation. The default threshold value is 0 and the range is 0 to 255.

**DTE1\_FSERR**

Received frame sync error

**DTE1\_FSERROK**

Received frame sync error recovered

**DTE1\_LOOPBACK\_CFA**

Diagnostic mode on the line trunk

**DTE1\_LOOPBACK\_CFAOK**

Diagnostic mode on the line trunk recovered

**DTE1\_LOS**

Received loss of signal

DTE1_LOSOK	Received loss of signal recovered
DTE1_MFSERR	Received multi-frame sync error
DTE1_MFSERROK	Received multi-frame sync error recovered
DTE1_RDMA	Received distant multi-frame alarm
DTE1_RDMAOK	Received distant multi-frame alarm recovered
DTE1_RED†	Received red alarm
DTE1_REDOK	Received red alarm recovered
DTE1_RLOS	Received loss of sync
DTE1_RLOSOK	Received loss of sync recovered
DTE1_RRA†	Received remote alarm
DTE1_RRAOK	Received remote alarm recovered
DTE1_RSA1	Received signaling all 1's
DTE1_RSA1OK	Received signaling all 1's recovered
DTE1_RUA1	Received unframed all 1's
DTE1_RUA1OK	Received unframed all 1's recovered

The following list shows the detected (incoming) alarms that are supported on T1 for DM3 boards. The dagger symbol (†) next to an alarm name indicates that the alarm is blocking by default. The default can be changed using **gc\_SetAlarmConfiguration( )**. For alarms where a default threshold value is shown, the default can be changed in the .config file for the board as explained in the DM3 Configuration Guide.

DTT1_BPVS	Bipolar violation count saturation. The default threshold value is 255 and the range is 0 to 255.
DTT1_ECS	Frame sync bit error count saturation. The default threshold value is 0 and the range is 0 to 255.

**DTT1\_FERR**

Two out of four consecutive frame bits (F bit) in error. The default threshold value is 0 and the range is 0 to 255.

**DTT1\_LOOPBACK\_CFA**

Diagnostic mode on the line trunk

**DTT1\_LOOPBACK\_CFAOK**

Diagnostic mode on the line trunk recovered

**DTT1\_LOS**

Initial loss of signal detected

**DTT1\_LOSOK**

Signal restored

**DTT1\_OOF**

Out of frame error count saturation. The default threshold value is 0 and the range is 0 to 255.

**DTT1\_RBL**

Received blue alarm

**DTT1\_RBLOK**

Received blue alarm restored

**DTT1\_RCL**

Received carrier loss

**DTT1\_RCLOK**

Received carrier loss restored

**DTT1\_RED†**

Received a red alarm condition

**DTT1\_REDOK**

Red alarm condition recovered

**DTT1\_RLOS**

Received loss of sync

**DTT1\_RLOSOK**

Received loss of sync restored

**DTT1\_RYEL†**

Received yellow alarm

**DTT1\_RYELOK**

Received yellow alarm restored

## 4.6.2 Alarm Handling for Springware Boards

As described in the *Global Call API Library Reference*, the GCEV\_BLOCKED event indicates that a line is blocked and the application cannot issue call-related function calls, and the GCEV\_UNBLOCKED event indicates that the line has become unblocked. For example, an alarm condition has occurred or has been cleared, respectively. These events are generated on every opened line device associated with the trunk on which the alarm occurs, if the event is enabled.

These events are enabled by default. The application may disable and enable the events by using the **gc\_SetEvtMsk()** function.

Setting the event mask on any line device that represents a time slot will result in setting the mask to the same value on all time slot level line devices on the same trunk. Additionally, setting the event mask on a line device that represents the board will have the same effect (that is, it will set the mask for all time slot level line devices on that trunk).

When an alarm occurs on a Global Call line device, the application must call the **dx\_stopch()** function to stop any application initiated voice processing, such as **dx\_play()** and **dx\_record()**, that is associated with that line device. The application should wait for the receipt of the GCEV\_UNBLOCKED event that signals the end of the alarm condition; then the application can proceed with its call processing (for example, making or receiving calls).

Alarm notification can be configured for time slot devices using the Global Call Alarm Management System (GCAMS). The Global Call functions that comprise the GCAMS interface for alarm management are supported. See the *Global Call API Programming Guide* for more information on GCAMS and the *Global Call API Library Reference* for more information on the GCAMS functions.

The **gc\_TransmitAlarms()** and **gc\_StopTransmitAlarms()** functions can be used to start and stop the transmission of alarms to the remote side. Table 9 gives the alarms that can be transmitted on E1 and T1 interfaces.

**Table 9. Alarms That Can Be Transmitted on E1 and T1 Interfaces on Springware Boards**

E1 Alarm	T1 Alarm	Description
DEA_REMOTE †	YELLOW †	Remote alarm indication (RAI)
DEA_UNFRAMED1 †	BLUE †	Alarm indication signal (AIS)
DEA_SIGNALALL1 †	—	Signaling all 1s alarm (a multi-frame alarm)
DEA_DISTANTMF †	—	Distant multi-frame alarm
† Defines that can be used in the alarm_number field of the ALARM_FIELD structure when using the <b>gc_TransmitAlarms()</b> and <b>gc_StopTransmitAlarms()</b> functions to start and stop the transmission of specific alarms.		

The following list shows the detected (incoming) alarms that are supported on E1 for Springware boards. The dagger symbol (†) next to an alarm name indicates that the alarm is blocking by default. The default can be changed using **gc\_SetAlarmConfiguration()**.

DTE1\_BPVS†  
Bipolar violation count saturation

DTE1\_BPVOK  
Bipolar violation count saturation recovered

DTE1\_CECS†  
CRC4 error count saturation

DTE1\_CECSOK  
CRC4 error count saturation recovered

DTE1\_DPM†  
Driver performance monitor failure

DTE1\_DPMOK  
Driver performance monitor failure recovered

DTE1\_ECS†  
Error count saturation

DTE1\_ECSOK  
Error count saturation recovered

DTE1\_FSERR†  
Received frame sync error

DTE1\_FSERROK  
Received frame sync error recovered

DTE1\_LOS†  
Received loss of signal

DTE1\_LOSOK  
Received loss of signal recovered

DTE1\_MFSERR†  
Received multi-frame sync error

DTE1\_MFSERROK  
Received multi-frame sync error recovered

DTE1\_RDMA†  
Received distant multi-frame alarm

DTE1\_RDMAOK  
Received distant multi-frame alarm recovered

DTE1\_RED  
Received red alarm

DTE1\_REDOK  
Received red alarm recovered

DTE1\_RLOS†  
Received loss of sync

DTE1\_RLOSOK  
Received loss of sync recovered

DTE1\_RRA†  
Received remote alarm

DTE1\_RRAOK  
Received remote alarm recovered

DTE1\_RSA1†  
Received signaling all 1's

DTE1\_RSA1OK  
Received signaling all 1's recovered

DTE1\_RUA1†  
Received unframed all 1's

DTE1\_RUA1OK  
Received unframed all 1's recovered

The following list shows the detected (incoming) alarms that are supported on T1 for Springware boards. The dagger symbol (†) next to an alarm name indicates that the alarm is blocking by default. The default can be changed using **gc\_SetAlarmConfiguration()**.

DTT1\_B8ZSD†  
Bipolar eight zero substitution detected

DTT1\_B8ZSD  
Bipolar eight zero substitution detected recovered

DTT1\_BPVS†  
Bipolar violation count saturation

DTT1\_BPVSOK  
BPVS restored

DTT1\_DPM†  
Driver performance monitor

DTT1\_DPMOK  
Driver performance monitor restored

DTT1\_ECS†  
Error count saturation

DTT1\_ECSOK  
Error count saturation recovered

DTT1\_FERR†  
Frame bit error

DTT1\_FERROK  
Frame bit error restored

DTT1\_LOS†  
Initial loss of signal detected

DTT1\_LOSOK  
Signal restored

DTT1\_OOF†  
Out of frame error; count saturation

DTT1\_OOFOK  
Out of frame restored

DTT1\_RBL†  
Received blue alarm

DTT1\_RBLOK  
Received blue alarm recovered

DTT1\_RCL†  
 Received carrier loss  
 DTT1\_RCLOCK  
 Received carrier loss restored  
 DTT1\_RED†  
 Received a red alarm condition  
 DTT1\_REDOK  
 Red alarm condition recovered  
 DTT1\_RLOS†  
 Received loss of sync  
 DTT1\_RLOSOK  
 Received loss of sync restored  
 DTT1\_RYEL†  
 Received yellow alarm  
 DTT1\_RYELOK  
 Received yellow alarm restored

## 4.7 Run Time Configuration of the PDKRT Call Control Library

**Note:** The information in this section is applicable to Springware boards only.

Table 10 shows the parameters of the PDKRT call control library that can be configured using the real time configuration management (RTCM) functions. The **gc\_GetConfigData()** function can be used to retrieve the target object configuration, and the **gc\_SetConfigData()** function can be used to update the target object configuration.

**Note:** Since these parameters are statically defined, the **gc\_QueryConfigData()** is not applicable.

**Table 10. Configurable PDKRT Call Control Library Parameters**

Set ID	Parm ID	Target Object Type	Description	Data Type	Access Attribute*
GCSET_CALLINFO	CALLINFO TYPE	GCTGT_CCLIB_CRN	Calling info type (alternative to <b>gc_GetCallInfo()</b> )	string	GC_R_O
	CATEGORY_DIGIT	GCTGT_CCLIB_CRN	Category digit type (alternative to <b>gc_GetCallInfo()</b> )	char	GC_R_O
*Access attributes are: GC_W_I: update GC_R_O: retrieve only GC_W_N: update only at null state GC_W_X: not available					



**Table 10. Configurable PDKRT Call Control Library Parameters (Continued)**

Set ID	Parm ID	Target Object Type	Description	Data Type	Access Attribute*
	CONNECT_ TYPE	GCTGT_CCLIB_ CRN	Connect type (alternative to <b>gc_GetCallInfo( )</b> )	char	GC_R_O
GCSET_ PARM	GCPR_ CALLING PARTY	GCTGT_CCLIB_ CHAN	Calling party (alternative to <b>gc_GetParm( )</b> / <b>gc_SetParm( )</b> )	string	GC_W_I
	GCPR_ LOADTONES	GCTGT_CCLIB_ CHAN	Load tones (alternative to <b>gc_GetParm( )</b> / <b>gc_SetParm( )</b> )	short	GC_W_I
	GCPR_ MEDIADetect	GCTGT_CCLIB_ CHAN	Set Media Detect (alternative to <b>gc_SetParm( )</b> )	short	GC_W_I
GCSET_ ORIG_ ADDR	GCPARM_ ADDR_DATA	GCTGT_CCLIB_ CHAN	Calling number (alternative to <b>gc_SetCallingNum( )</b> )	string	GC_W_I
*Access attributes are: GC_W_I: update GC_R_O: retrieve only GC_W_N: update only at null state GC_W_X: not available					

## 4.8 Run Time Configuration of PDK Protocol Parameters

**Note:** The information in this section is applicable to Springware boards only.

Configurable PDK protocol parameters are grouped into two sets:

- Protocol state information (PSI) variable parameters
- Protocol service layer (PSL) variable parameters

**Note:** To avoid errors, the PSI and PSL parameters of a GCTGT\_PROTOCOL\_CHAN channel are allowed to be changed only when the channel object does not have an active call.

PSI variable parameters are interpreted by the PDK run-time component (PDKRT). The names of the PSI variable parameters (beginning with CDP\_) are found in the .cdp file. The PSI parameters that can be accessed via **gc\_GetConfigData( )**, **gc\_SetConfigData( )**, and **gc\_QueryConfigData( )** are protocol dependent. Refer to the *Global Call Country Dependent Parameters (CDP) for PDK Protocols Configuration Guide* for further information.

The PSL variable parameters are not available to the protocol state machine, but rather are used by the protocol services layer to control the behavior of various network and voice functions. The names of the PSL variable parameters begin with PSL\_ and SYS\_. No variation in the names is allowed. These parameters are required to control protocol parameters (e.g., timing) or they may control the behavior of the underlying implementation. In the latter case, the parameters will most

likely have a platform tag. All of these parameter names must begin with PSL. The PSL parameters that can be accessed via **gc\_GetConfigData()**, **gc\_SetConfigData()**, and **gc\_QueryConfigData()** are shown in Table 11.

**Table 11. PSL and SYS Parameters**

PSL Variable Name	Data Type
PSL_AcceptCallDefaultNumOfRings	Integer
PSL_AnswerCallDefaultNumOfRings	Integer
PSL_MakeCall_CallProgress	Integer
PSL_MakeCall_MediaDetect	Integer
PSL_DefaultMakeCallTimeout	Integer
PSL_DXCAS_HOOKFLASH_DURATION	Integer
SYS_FEATURES	String
SYS_PSINAME	String

Table 12 shows the Set ID and Parm ID for these parameter types.

**Table 12. Configurable PDK Protocol Parameters**

Set ID	Parm ID	Target Object Type	Explanation	Access Attribute**
PDKSET_PSI_VAR *	Dynamically assigned	GCTGT_PROTOCOL_SYSTEM, GCTGT_PROTOCOL_CHAN	Protocol state information (PSI) variable parameters	GC_W_N
PDKSET_SERVICE_VAR	Dynamically assigned	GCTGT_PROTOCOL_SYSTEM, GCTGT_PROTOCOL_CHAN	Protocol service layer (PSL) variable parameter and system parameters	GC_W_N
*Indicates that CAS pattern signals and tones cannot be accessed. **Access attributes are: GC_W_I: update GC_R_O: retrieve only GC_W_N: update only at null state GC_W_X: not available				

The PDK GCTGT\_PROTOCOL\_SYSTEM target object is not available until the first **gc\_OpenEx()** function is called to run this protocol.

The Global Call application can call **gc\_GetConfigData()** to retrieve protocol configuration information or **gc\_SetConfigData()** to set protocol configuration information. Since these parameters are protocol dependent, their parameters are dynamically assigned when a protocol is loaded into the PDKRT. Therefore, a Global Call application must call **gc\_QueryConfigData()** to find the parameter information (set ID, parm ID, and value data type, etc.) first. For more information about these functions, refer to the *Global Call API Programming Guide*.

The pair (target object type, target object ID) supporting **gc\_QueryConfigData()** to find PDKRT protocol parameter information can be one of the following:

- (GCTGT\_PROTOCOL\_SYSTEM, Global Call protocol ID)
- (GCTGT\_PROTOCOL\_CHAN, Global Call line device ID)

For a given protocol, although the GCTGT\_PROTOCOL\_SYSTEM target object and GCTGT\_PROTOCOL\_CHAN target object share the same set ID and parm ID for PSI variables, they can have different values. When a new GCTGT\_PROTOCOL\_CHAN target object is opened, it gets a copy of the current PSI variable configuration of GCTGT\_PROTOCOL\_SYSTEM target object. Under this situation, changes to the GCTGT\_PROTOCOL\_SYSTEM target object configuration will not affect the configuration of the GCTGT\_PROTOCOL\_CHAN target object. But the GCTGT\_PROTOCOL\_SYSTEM target object shares the same PSL variable configuration with other GCTGT\_PROTOCOL\_CHAN target objects.

The following example shows how to set the **CDP\_ANI\_ENABLED** parameter for channel ldev running a PDK protocol at the NULL state in asynchronous mode.

**Note:** Error handling is not shown.

```
GC_PARM t_SourceParm, t_DestParm;
GC_PARM_ID t_ParmIDSt;
char t_name[20] = "CDP_ANI_ENABLED";
long request_id;
LINEDEV ldev;
GC_PARM_BLK * t_pParmBlk = NULL;

/* first find the parameter info by calling gc_QueryConfigData() function */
t_SourceParm.paddress = t_name; /* Pass the PSI variable name */
memset(&t_ParmIDSt, 0, sizeof(GC_PARM_ID));
t_DestParm.pstruct = &t_ParmIDSt; /* Pass desired the parm info */
gc_QueryConfigData(GCTGT_PROTOCOL_CHAN, ldev, &t_SourceParm,
                  GCQUERY_PARM_NAME_TO_ID, &t_DestParm);

/* Call GC utility function to insert a parameter data to GC_PARM_BLK */
gc_util_insert_parm_val(&t_pParmBlk, t_ParmIDSt.set_ID,
                      t_ParmIDSt.parm_ID, sizeof(int), 10);

/* Call gc_SetConfigData() function to set the "CDP_ANI_ENABLE" */
gc_SetConfigData(GCTGT_PROTOCOL_CHAN, ldev, t_pParmBlk, 0,
                GCUPDATE_ATNULL, &request_id, EV_ASYNC);
...
/* Call GC utility function to release the memory after using the GC_PARM_BLK */
gc_util_delete_parm_blk(t_pParmBlk);
```

## 4.9 Determining the Protocol Version

**Note:** The information in this section is applicable to Springware boards only.

The following software code demonstrates how you can determine the Global Call protocol version you are running.

```
#include <gclib.h>
#include <gcerr.h>
#include <srllib.h>
```

```

int main()
{
    LINEDEV    ldev;
    GC_PARM    parm;
    int        retcode;
    METAEVENT  metaevent;
    parm.paddress = NULL;
    int        mode;

#ifdef _WIN32
    mode = SR_STASYNC|SR_POLLMODE;
#else
    mode = SR__POLLMODE;
#endif

    if (sr_setparm(SRL_DEVICE, SR_MODELTYPE, &mode) == -1)
    {
        // Error processing
    }

    gc_Start(NULL);
    retcode = gc_Open(&ldev, "P_na_an_io:N_dtiB1t1:V_dxxxB1C1", 0);
    if (retcode != GC_SUCCESS)
    {
        // Error processing
    }

    sr_waitevt(50);
    retcode = gc_GetMetaEvent(&metaevent);
    if (retcode != GC_SUCCESS)
    {
        // Error processing
    }
    if (metaevent.flags & GCME_GC_EVENT)
    {
        if (metaevent.evtttype == GCEV_UNBLOCKED)
        {
            if (gc_GetParm(ldev, GCPR_PROTVER, &parm) == GC_SUCCESS)
            {
                printf("The protocol version: %s\n", parm.paddress);
            }
            else
            {
                // Error processing
            }
        }
    }

    gc_Close(ldev);
    gc_Stop();
    return(0);
}

```

This chapter describes the E1/T1 CAS/R2 protocols supported by Global Call. Topics include:

- [Protocols Supported](#) . . . . . 61
- [Protocol File Naming Conventions](#) . . . . . 61
- [Protocol Components](#) . . . . . 63

- Notes:**
1. See the *Global Call Country Dependent Parameters (CDP) for PDK Protocols Configuration Guide* for more information about using the protocols and the country dependent parameter (CDP) files, including detailed procedures for configuring country dependent parameters and for downloading the protocol and CDP file.
  2. With newer releases of Intel® Dialogic® system release software, the Global Call Protocols can now be installed as part of the system release or with a Service Update for the system release. You do not have to install the Global Call Protocols separately as in the past.
  3. The development of the ICAPI protocols supported by Global Call has been capped. Customers should migrate to equivalent protocols developed using the Protocol Development Kit (PDK). New protocol development as well as existing protocol support will be on the PDK. ICAPI protocols are supported only on Springware boards. PDK protocols are supported on both DM3 boards and Springware boards.

## 5.1 Protocols Supported

The Global Call protocols available are listed in the *Global Call Country Dependent Parameters (CDP) for PDK Protocols Configuration Guide*. For the most up-to-date list of available protocols, contact your nearest Intel Sales Office.

The protocol and parameters used at the application's interface to the PTT must complement those used by the local CO. To maintain compatibility with the local PTT, Intel provides .cdp country dependent parameter files that can be modified to satisfy local requirements. User selectable options allow customization of the country dependent parameters to fit a particular application or configuration within a country (for example, switches within the same country may use the same protocol but may require different parameter values for local use). These parameters (for example, the number of DNIS digits, time-outs, party calling number, idle patterns, signaling patterns, and protocol-specific definitions) are specified in the .cdp file and may be modified at configuration time (that is, at any time before starting your application). See the *Global Call Country Dependent Parameters (CDP) for PDK Protocols Configuration Guide* for additional information.

## 5.2 Protocol File Naming Conventions

When a protocol is installed on your system, several files are installed, including the protocol module(s) and country dependent parameter files. For most protocols, the files are named according to the conventions in Table 13.

Table 13. Protocol File Naming Conventions

File Name	Description
<i>ccl_cc_tt_d.hot</i> and <i>ccl_cc_tt_d.qs</i>	PDK protocol module (DM3)
<i>ccl_cc_tt_d.psi</i>	PDK protocol module (Springware)
<i>ccl_cc_tt_d.so</i> or <i>ccl_cc_tt_ffff_d.so</i>	ICAPI protocol module for Linux (Springware only)
<i>ccl_cc_tt_d.dll</i> or <i>ccl_cc_tt_ffff_d.dll</i>	ICAPI protocol module for Windows (Springware only)
<i>ccl_cc_tt_d.cdp</i> or <i>ccl_cc_tt_ffff_d.cdp</i>	Country dependent parameter file (DM3 and Springware)

where:

- **ccl** indicates the call control library for which the protocol is written, for example, **pd**k represents the PDKRT call control library. For the ICAPI call control library, **ccl** is blank.
- **cc** is a 2-character ISO country code, regional code (for example, **es** = Spain, **fr** = France, **mx** = Mexico, **na** = North America, etc.), or an indication of a switch-specific protocol.
- **tt** is a 2-character protocol type. Valid types are:
  - **em**: a T1 protocol using E&M signaling with support for DTMF digits only
  - **mf**: a T1 protocol using E&M signaling with support for MF digits
  - **r2**: a protocol using R2 MFC signaling
  - **r1**: a protocol using R1 MFC signaling
  - **e1**: a pulse, MF SOCOTEL, or other E1 protocol
  - **sw**: a protocol that is switch specific
  - **ls**: a loop start protocol
- **d** is a 1- or 2-character direction indicator. Valid directions are:
  - **i**: inbound
  - **o**: outbound
  - **io**: inbound/outbound
- **ffff** is optional and defines a special software or hardware feature supported by the protocol; 1 to 4 characters. If the protocol type is “sw”, then this field provides additional information about the switch.
 

**Note:** Requires ICAPI call control library level 2, or else a compatibility error, EGC\_COMPATIBILITY, will be generated when the application attempts to load the protocol.

The protocol name used in the **devicename** parameter of the **gc\_OpenEx()** function is the root name of the .cdp file. (On DM3 boards, the protocol is determined at board initialization time and not when a Global Call device is opened. For compatibility, the **gc\_OpenEx()** protocol name may be specified for DM3 boards, but it is not used.)

Most ICAPI protocol releases use separate protocol modules to handle the inbound and the outbound portions of a protocol. For example, Table 14 describes the files included for the Argentina R2 ICAPI protocol.

**Table 14. Sample ICAPI Protocol File Set**

Description	Protocol Files	
	Linux	Windows
Inbound protocol module	<i>ar_r2_i.so</i>	<i>ar_r2_i.dll</i>
Outbound protocol module	<i>ar_r2_o.so</i>	<i>ar_r2_o.dll</i>
Inbound country dependent parameters	<i>ar_r2_i.cdp</i>	<i>ar_r2_i.cdp</i>
Outbound country dependent parameters	<i>ar_r2_o.cdp</i>	<i>ar_r2_o.cdp</i>

PDK protocols are bidirectional protocols. For example, Table 15 describes the files included with the Argentina R2 PDK protocol.

**Table 15. Sample PDK Protocol File Set**

Description	Protocol Files
	Linux and Windows
Bidirectional protocol module (DM3)	<i>pdk_r2_io.hot</i> , <i>pdk_r2_io.qs</i>
Bidirectional protocol module (Springware)	<i>pdk_r2_io.psi</i>
Bidirectional country dependent parameters (DM3 and Springware)	<i>pdk_ar_r2_io.cdp</i>

## 5.3 Protocol Components

Each protocol requires specific firmware parameter file(s) to be downloaded to the voice and network boards:

- [Protocol Modules](#)
- [Country Dependent Parameter \(.cdp\) Files](#)

### 5.3.1 Protocol Modules

These files contain protocol specific information and are dynamically linked to the application as needed.

PDK protocols are supported on both DM3 and Springware boards. For DM3 boards, the protocol modules are .hot and .qs files. For Springware boards, the protocol module is a protocol state information (.psi) file, a binary file that is interpreted by the PDK run-time component (PDKRT).

ICAPI protocols are supported on Springware boards only. The protocol modules for Linux are .so files. The protocol modules for Windows are .dll files.

### 5.3.2 Country Dependent Parameter (.cdp) Files

These files contain country specific and protocol specific parameters for use by Global Call. Country dependent parameter (.cdp) files may be customized. Descriptions of the country dependent parameters most likely to be modified for a protocol are provided in the *Global Call Country Dependent Parameters (CDP) for PDK Protocols Configuration Guide*.

For ICAPI protocols, the special parameter @0 identifies the protocol to be run. This parameter specifies the name of the protocol module (ignoring the filename extension and without the path) to be run by the application. Two variations of the same protocol can be run if two .cdp files point to the same protocol module filename after @0.

The .cdp file should be located only under the installation directory:

- For Linux: \$INTEL\_DIALOGIC\_CFG
- For Windows: %INTEL\_DIALOGIC\_CFG%



# Building Global Call E1/T1 CAS/R2 Applications

---

## 6

This chapter describes the E1/T1 CAS/R2 specific header files and libraries required when building applications.

- Header Files . . . . . 65
- Required Libraries . . . . . 65
- Required System Software . . . . . 65

### 6.1 Header Files

When compiling Global Call applications for the E1/T1 CAS/R2 technology, it may be necessary to include the following header files in addition to the standard Global Call header files, which are listed in the *Global Call API Library Reference* and *Global Call API Programming Guide*:

#### For DM3 Boards

*dm3cc\_parm.h*  
required when using DM3 boards

#### For Springware Boards

*gcpdkrt.h*  
required when using PDK error codes, the PDK\_MAKECALL\_BLK structure for call analysis, or logging via the **gc\_Start()** function

*icapi.h*  
required when using ICAPI error codes and features

### 6.2 Required Libraries

When building Global Call applications for the E1/T1 CAS/R2 technology, it is not necessary to link any libraries other than the standard Global Call library, *libgc.lib*.

### 6.3 Required System Software

The Intel® Dialogic® system software must be installed on the development system. See the Software Installation Guide for your system release for further information.



# Debugging Global Call E1/T1 CAS/R2 Applications

---

7

The Global Call debugging utilities are described in this chapter.

- [Introduction](#) ..... 67
- [Debugging Applications that Use PDK Protocols](#)..... 67
- [Debugging Applications that Use ICAPI Protocols](#) ..... 72

**Note:** The information in this chapter is applicable to Springware boards only. For information about the pdktrace tool used with DM3 boards, see the Diagnostics Guide for your system release. The pdktrace tool requires Global Call Protocols Version 4.1 or later.

## 7.1 Introduction

Global Call includes powerful debugging capabilities for troubleshooting protocol-related problems, including the ability to generate a detailed log file. These debugging tools should not be used during normal operations or when running an application for an extended period of time since they increase the processing load on the system and they can quickly generate a large log file.

**Note:** Only run the debugging and logging utilities on a limited number of channels at a time to avoid the possibility of losing events.

## 7.2 Debugging Applications that Use PDK Protocols

This section discusses the following topics:

- [Enabling and Disabling the Logging](#)
- [Populating and Using a CCLIB\\_START\\_STRUCT](#)
- [Defining the GC\\_PDK\\_START\\_LOG Environment Variable](#)

### 7.2.1 Enabling and Disabling the Logging

The Global Call PDKRT (Protocol Development Kit Run Time) provides a rich set of logging features that are useful to protocol developers and implementers of the engine and call control libraries. The application may add additional log records to the log file when logging is enabled.

- Notes:**
1. It is recommended to use logging on an as-needed basis. Logging uses significant resources and can reduce the performance of the Global Call PDKRT call control library. Full logging (debug logging) enabled on many channels can reduce performance to such a degree that time-critical operations are affected and the behavior of a protocol may be altered.
  2. The LogView tool is required to view the log file.

The PDKRT call control library provides a service for capturing error and debug information in a log file. Enabling and disabling logging is achieved using the **gc\_Start()** function. Once logging is enabled, the **gc\_StartTrace()** function can be used to enable logging on each individual channel. See [Section 8.2.22, “gc\\_Start\(\) and gc\\_Stop\(\) Variances for E1/T1 CAS/R2”](#), on page 94 and [Section 8.2.23, “gc\\_StartTrace\(\) Variances for E1/T1 CAS/R2”](#), on page 94 for more information.

The parameters that control the logging mechanism can be set by:

- Populating and using a CCLIB\_START\_STRUCT. See [Section 7.2.2, “Populating and Using a CCLIB\\_START\\_STRUCT”](#), on page 68.
- Defining the GC\_PDK\_START\_LOG environment variable. See [Section 7.2.3, “Defining the GC\\_PDK\\_START\\_LOG Environment Variable”](#), on page 72.

When both methods are used, the CCLIB\_START\_STRUCT takes precedence over the GC\_PDK\_START\_LOG environment variable.

**Note:** Two applications should not use the same log file.

## 7.2.2 Populating and Using a CCLIB\_START\_STRUCT

The following code shows an example of how to define a CCLIB\_START\_STRUCT, populate the fields, and use it to enable logging when issuing the **gc\_Start()** function.

```
GC_START_STRUCT t_GcStart;
CCLIB_START_STRUCT t_PdkStart;
t_PdkStart.cclib_name = "GC_PDKRT_LIB";
t_PdkStart.cclib_data = "filename: pdktest.log;
loglevel: ENABLE_DEBUG;
service: R2MF_ENABLE | CAS_ENABLE;
cachedump: WHEN_FULL | THREAD_ON;
channel: B1C1, B2C2-4;
cachesize: 10;
maxfilesize: 0;
mindiskfree: 20";
t_GcStart.num_cclibs = 1;
t_GcStart.cclib_list = (void *)
    (& t_PdkStart);
int t_result = gc_Start((GC_START_STRUCTP)& t_GcStart);
```

**Note:** The example above shows all the possible fields in a **cclib\_data** string. In practice, you only need to specify the values of fields that are different than the default values.

The length of the filename must be less than 8 characters.

The value of the **cclib\_name** field must be GC\_PDKRT\_LIB and the **cclib\_data** field should have the following format:

```
"field name 1 : field value 1; field name 2 : field value 2; ..."
```

where the allowable field names and values are given in Table 16.

**Table 16. cclib\_data Fields and Values**

Field Name	Field Values	Default Value
filename	Log file name	gc_pdk.log
loglevel	See Table 17.	ENABLE_FATAL or 5
service	See Table 18.	ALL_SERVICES
cachedump	See Table 19.	WHEN_FULL or 1
cachesize	Any positive integer	1 (number of records in cache)
channel	See Table 20.	B*C*
maxfilesize	Integer	0 (Megabytes)
mindiskfree	Integer	20 (Megabytes)

The fields can be defined in any sequence. If any field is not defined or defined incorrectly (either in name or value), then the default value is used for logging. The actual values of the fields are posted as the first record of the log file. In this way, when a log file is received, the user knows how logging was configured (that is, which log level and services were enabled, what the cache size and cache dump conditions were when it was generated).

The following examples show how to set the **cclib\_data** string:

- The example below shows all the possible fields. In practice, you only have to specify the values of fields that are different than the default values.

```
cclib_data = "filename: pdktest.log;
loglevel: ENABLE_DEBUG;
service: R2MF_ENABLE;
cachedump: WHEN_FULL|THREAD_ON;
channel: B1C1, B2C2-4;
cachesize: 10;
maxfilesize: 0;
mindiskfree: 20"
```

- For simplicity and to avoid errors, use only the values of fields that are different than the default values. For example, to specify a log file name called *mylog.log* that includes all log entries, use the following **cclib\_data** string:

```
cclib_data = "filename: mylog.log; loglevel: ENABLE_DEBUG"
```

The following tables show the allowable values for the **loglevel**, **service**, **cachedump**, and **channel** fields respectively. The values of **loglevel**, **service**, and **cachedump** can be numbers or symbols. (If hex format is used, the prefix 0x should be used.) Consequently, before these values are passed to the LOG\_INIT, the values must be examined and converted from symbols to numbers, if necessary. The value symbol of **service** and **cachedump** can be a bit mask.

Table 17 shows the valid values for the **loglevel** parameter.

Table 17. Loglevel Parameter Values

loglevel	Valid Value	Description
ENABLE_FATAL (default)	5	Only fatal errors are logged. A fatal error is an error that will make the program run abnormally or will stop the program. For example, in <i>channelimpl.cpp</i> , <b>dx_open( )</b> returns INVALID_VOICEH. It is expected that an exception will be thrown and the log cache will be dumped to a file if possible.
ENABLE_WARNING	4	All levels above ALERT are logged. An error occurs that may make the program run abnormally. For example, in <i>channelimpl.cpp</i> , the new local state is not ChanState_InService while the reason is Wait Call. An exception may be thrown, but log cache will not be dumped to a file automatically.
ENABLE_ALERT	3	All levels above INFO are logged. There is a problem, generally not an error, that the user should know about.
ENABLE_INFO	2	All levels above DEBUG are logged. Important information that the user needs to be aware of is logged. For example, in <i>channelimpl.cpp</i> , issuing a <b>gc_StartTrace( )</b> and <b>gc_StopTrace( )</b> determines if logging for a specific channel is on or off. This kind of information is a level higher than DEBUG.
ENABLE_DEBUG	1	All levels are logged. This gives the most detailed information to help debug protocols or code step-by-step. For example, in <i>channelimpl.cpp</i> , a call to any of the GC_PDK_C_XXX functions should be logged at this level. Most routine logging should use this level.
<b>Note:</b> Values are in decimal but can also be specified in hex using a 0x prefix.		

Table 18 shows the valid values for the **service** parameter.

Table 18. Service Parameter Values

service	Valid Value	Description
ALL_SERVICES (default)	0xFFFFFFFF (65535)	All services are enabled.
USRAPP_ENABLE	0x00000001 (1)	Only USRAPP service enabled.
GCAPI_ENABLE	0x00000002 (2)	Only GCAPI service enabled.
GCXLTR_ENABLE	0x00000004 (4)	Only GCXLTR service enabled.
LINEADMIN_ENABLE	0x00000008 (8)	Only LINEADMIN service enabled.
CHANNEL_ENABLE	0x00000010 (16)	Only CHANNEL service enabled.
LOADER_ENABLE	0x00000020 (32)	Only LOADER service enabled.
CALL_ENABLE	0x00000040 (64)	Only CALL service enabled.
R2MF_ENABLE	0x00000080 (128)	Only R2 MF service enabled.
TONE_ENABLE	0x00000100 (256)	Only TONE service enabled.
CAS_ENABLE	0x00000200 (512)	Only CAS service enabled.
TIMER_ENABLE	0x00000400 (1024)	Only TIMER service enabled.
<b>Note:</b> Values prefixed with 0x are hexadecimal values. Decimal values are shown in parentheses.		

**Table 18. Service Parameter Values (Continued)**

service	Valid Value	Description
SDL_ENABLE	0x00000800 (2048)	Only SDL service enabled.
SRL_ENABLE	0x00001000 (4096)	Only SRL service enabled.
ERRHNDLR_ENABLE	0x00002000 (8192)	Only ERRHNDLR service enabled.
LOGGER_ENABLE	0x00004000 (16384)	Only LOGGER service enabled.
RTCM_ENABLE	0x00008000 (32768)	Only RTCM service enabled.
GCLIB_ENABLE	0x00010000 (65536)	Only GCLIB service enabled.
<b>Note:</b> Values prefixed with 0x are hexadecimal values. Decimal values are shown in parentheses.		

Table 19 shows the valid values for the **cachedump** parameter.

**Table 19. Cachedump Parameter Values**

cachedump	Valid Value	Description
ON_FATAL	0x0000 (bit 1 = 0)	The cache memory will be dumped to the log file once there is a log record with a FATAL level.
WHEN_FULL (default)	0x0001 (bit 1 = 1)	The cache memory will be dumped to the log file once the log cache is full as determined by the <b>cachesize</b> parameter. For example, if <b>cachesize</b> is 10, the log cache is dumped to a file when it contains 10 log records.
THREAD_OFF (default)	0x0000 (bit 2 = 0)	The dump operation will be executed by the calling thread.
THREAD_ON	0x0002 (bit 2 = 1)	The dump operation will be executed by a separate cache dumping thread.
<b>Note:</b> Values prefixed with 0x are hexadecimal values.		

Table 20 shows some examples of the **channel** parameter.

**Table 20. Sample Channel Parameter Values**

Example Value	Boards and Channels Enabled for Logging
B*C* (default)	All boards and all channels
B-1C-1	Only board number = -1 and channel number = -1.
B1C*	All channels on board 1.
B1C-1	Only board 1 level.
B1C1	Channel 1 on board 1.
B1C1-5	Channels 1 to 5 on board 1.
B1C1,20	Channels 1 and 20 on board 1.
B1-4C*	All channels of boards 1 to 4.
B1C2, B2C2,20-22	Channel 2 on board 1, channels 2, 20, 21, and 22 on board 2.

### 7.2.3 Defining the GC\_PDK\_START\_LOG Environment Variable

The GC\_PDK\_START\_LOG environment variable can also be used to enable and configure logging.

The following examples show how to set the GC\_PDK\_START\_LOG environment variable in Windows:

- The following is an example of a GC\_PDK\_START\_LOG environment variable definition showing all the possible field values in the environment variable. In practice, you only have to specify the values of fields that are different than the default values.

```
set GC_PDK_START_LOG="filename : pdktest.log;
loglevel: ENABLE_DEBUG; services: ALL_SERVICES;
cachedump : WHEN_FULL | THREAD_ON; channel : B1C1, B2C2-4;
cachesize : 10; maxfilesize : 0; mindiskfree : 20"
```

- For simplicity and to avoid errors, use only the values of fields that are different than the default values. For example, to specify a log file name called *mylog.log* that includes all log entries, use the following GC\_PDK\_START\_LOG environment variable definition:

```
set GC_PDK_START_LOG = "filename: mylog.log; loglevel: ENABLE_DEBUG"
```

This definition is equivalent to the logging configuration used in [Section 7.2.2, “Populating and Using a CCLIB\\_START\\_STRUCT”](#), on page 68 and the definition for each field is also the same as described in that section.

The setting of the environment variable to enable PDK logging in Linux is:

```
export GC_PDK_START_LOG="filename:gc_pdk.log;loglevel:ENABLE_DEBUG;
service:ALL_SERVICES;cachedump:WHEN_FULL|THREAD_OFF;cachesize:1;maxfilesize:2"
```

## 7.3 Debugging Applications that Use ICAP1 Protocols

The parameters shown in Table 21 are available in the *icapi.cfg* file as debugging tools. Unless otherwise instructed, these parameters should retain their original settings.

The *icapi.cfg* file is located in the following directory:

- For Linux: \$INTEL\_DIALOGIC\_CFG
- For Windows: %INTEL\_DIALOGIC\_CFG%

When logging is enabled, the log file generated is *icapi.log.<pid>*, where pid = the process identification number.

For Linux applications, the log file is generated by compiling the *country.c* file with the symbol **DEBUG** defined and then setting the parameters \$11 and \$12 in the *icapi.cfg* file as indicated in the following table. To write additional information directly to the ICAP1 log file, use the **rs\_log\_printf()** function. This function works like the **fprintf()** function except that a file descriptor is not used.

For Windows applications, the log file is generated by setting parameters \$11 and \$12 in the *icapi.cfg* file as indicated in Table 21.



**Table 21. icapi.cfg File Parameters**

Parameter	Description
\$11	<p>Logging utility (default = 0):</p> <ul style="list-style-type: none"> <li>Set to 0 to ignore parameters \$12, \$13 and \$15.</li> <li>Set to 1 to enable logging, either to the screen (set \$13 parameter to 1) or to the <i>icapi.log.&lt;pid&gt;</i> file to track all the events that occur at the device selected for monitoring (parameter \$12). This setting enables the debug tools associated with the protocol. These tools help to locate the source of a protocol problem.</li> <li>(Windows only) Set to 2 to enable logging to a memory buffer and to generate an <i>icapi.inf</i> file. The <i>icapi.inf</i> file contains the memory address where the debug information is stored.</li> </ul> <p><b>Note:</b> Enabling logging is not recommended during normal operation due to the increased host processor loading.</p>
\$12	<p>Number of the channel to be monitored (default = 0):</p> <ul style="list-style-type: none"> <li>A value of 0 means monitor all opened devices.</li> <li>A value of -1 means do not monitor any device.</li> <li>Entering a channel number designates the channel to be monitored.</li> </ul>
\$13	<p>Echo on screen (default = 0):</p> <ul style="list-style-type: none"> <li>Set to 0 to ignore parameter.</li> <li>Set to 1 to send the debug information to the screen.</li> </ul>
\$14	<p>Disable DTI Wait Call function (default = 0):</p> <ul style="list-style-type: none"> <li>The 0 default value causes the DTI Wait Call firmware function to wait for an incoming call at the board firmware level.</li> <li>A value of 1 causes the DTI Wait Call firmware function to wait for an incoming call at the ICAPI call control library level.</li> </ul> <p>The value selected is protocol-dependent; do not change the default value unless instructed to do so in the documentation for your protocol.</p>
\$15	<p>(Linux only) Size of debug memory (default = 1; that is, 1 = 1 event or action in memory)</p> <p>The debug memory saves passed actions or events to a buffer. The built-in debug function does not use this feature. Change this parameter only if you implement your own debug function and you need a larger circular buffer than 1 event or action.</p> <ul style="list-style-type: none"> <li>Set to 1 to store one action or event in the buffer.</li> <li>Set to 0 to ignore feature (default).</li> </ul>
\$18	<p>Enables cadenced tones, such as ringback and busy, to be played using the firmware rather than using host-based function calls such as <b>dx_playtone( )</b> and <b>sleep( )</b>.</p> <ul style="list-style-type: none"> <li>Set to 0 to disable firmware cadence tones (default).</li> <li>Set to 1 to enable firmware cadence tones.</li> </ul>

Any unspecified parameter defaults to 0. If parameters \$13 and \$15 are set to 0, they are ignored.

Parameters \$16 and \$17 (not shown in Table 21) are for backwards compatibility only and should not be changed.



# *E1/T1 CAS/R2-Specific Function Information*

## 8

This chapter describes the Global Call API functions that have additional functionality or perform differently when used with E1/T1 CAS/R2 technology. The function descriptions are presented alphabetically and contain information that is specific to E1/T1 CAS/R2 applications. Generic function description information (that is, information that is not technology-specific) is provided in the *Global Call API Library Reference*.

Topics in this chapter include:

- [Global Call Functions Supported by E1/T1 CAS/R2 . . . . . 75](#)
- [Global Call Function Variances for E1/T1 CAS/R2 . . . . . 82](#)

## 8.1 Global Call Functions Supported by E1/T1 CAS/R2

The following is a full list of the Global Call functions that indicates the level of support when used with E1/T1 CAS/R2 technology. The list indicates whether the function is supported, not supported, or supported with variances.

### **gc\_AcceptCall()**

Supported with variances described in [Section 8.2.1, “gc\\_AcceptCall\(\) Variances for E1/T1 CAS/R2”](#), on page 83.

### **gc\_AcceptInitXfer()**

Not supported.

### **gc\_AcceptModifyCall()**

Not supported.

### **gc\_AcceptXfer()**

Not supported.

### **gc\_AlarmName()**

Supported.

### **gc\_AlarmNumber()**

Supported.

### **gc\_AlarmNumberToName()**

Supported.

### **gc\_AlarmSourceObjectID()**

Supported.

### **gc\_AlarmSourceObjectIDToName()**

Supported.

**gc\_AlarmSourceObjectName()**

Supported.

**gc\_AlarmSourceObjectNameToID()**

Supported.

**gc\_AnswerCall()**Supported with variances described in [Section 8.2.2, “gc\\_AnswerCall\(\) Variances for E1/T1 CAS/R2”](#), on page 83.**gc\_Attach()** (deprecated)

Supported.

**gc\_AttachResource()**

Supported.

**gc\_BlindTransfer()**Supported with variances described in [Section 8.2.3, “gc\\_BlindTransfer\(\) Variances for E1/T1 CAS/R2”](#), on page 83.**gc\_CallAck()**For Springware boards: Supported with variances described in [Section 8.2.4, “gc\\_CallAck\(\) Variances for E1/T1 CAS/R2”](#), on page 84. For DM3 boards: Not supported.**gc\_CallProgress()**

Not supported.

**gc\_CCLibIDToName()**

Supported.

**gc\_CCLibNameToID()**

Supported.

**gc\_CCLibStatus()** (deprecated)

Supported.

**gc\_CCLibStatusAll()** (deprecated)

Supported.

**gc\_CCLibStatusEx()**

Supported.

**gc\_Close()**Supported with variances described in [Section 8.2.5, “gc\\_Close\(\) Variances for E1/T1 CAS/R2”](#), on page 84.**gc\_CompleteTransfer()**Supported with variances described in [Section 8.2.6, “gc\\_CompleteTransfer\(\) Variances for E1/T1 CAS/R2”](#), on page 85.**gc\_CRN2LineDev()**

Supported.

**gc\_Detach()**Supported with variances described in [Section 8.2.7, “gc\\_Detach\(\) Variances for E1/T1 CAS/R2”](#), on page 85.

**gc\_DropCall()**

Supported with variances described in [Section 8.2.8, “gc\\_DropCall\(\) Variances for E1/T1 CAS/R2”](#), on page 85.

**gc\_ErrorInfo()**

Supported.

**gc\_ErrorValue()** (deprecated)

Supported.

**gc\_Extension()**

Supported with variances described in [Section 8.2.9, “gc\\_Extension\(\) Variances for E1/T1 CAS/R2”](#), on page 86.

**gc\_GetAlarmConfiguration()**

Supported.

**gc\_GetAlarmFlow()**

Supported.

**gc\_GetAlarmParm()**

For Springware boards: Supported. For DM3 boards: Not supported.

**gc\_GetAlarmSourceObjectList()**

Supported.

**gc\_GetAlarmSourceObjectNetworkID()**

Supported.

**gc\_GetANI()** (deprecated)

Supported.

**gc\_GetBilling()**

Not supported.

**gc\_GetCallInfo()**

Supported with variances described in [Section 8.2.10, “gc\\_GetCallInfo\(\) Variances for E1/T1 CAS/R2”](#), on page 86.

**gc\_GetCallProgressParm()**

For Springware boards: Supported (PDKRT only). For DM3 boards: Not supported.

**gc\_GetCallState()**

Supported.

**gc\_GetConfigData()**

For Springware boards: Supported (PDKRT only). For DM3 boards: Not supported.

**gc\_GetCRN()**

Supported.

**gc\_GetCTInfo()**

For Springware boards: Supported (PDKRT only). For DM3 boards: Supported.

**gc\_GetDNIS()** (deprecated)

Supported.

**gc\_GetFrame()**

Not supported.

**gc\_GetInfoElem()**

Not supported.

**gc\_GetLineDev()**

Supported.

**gc\_GetLineDevState()**

For Springware boards: Not supported. For DM3 boards: Supported.

**gc\_GetMetaEvent()**

Supported.

**gc\_GetMetaEventEx()**

Supported (Windows\* extended asynchronous mode only).

**gc\_GetNetCRV()**

Not supported.

**gc\_GetNetworkH()** (deprecated)

Supported.

**gc\_GetParm()**Supported with variances described in [Section 8.2.11, “gc\\_GetParm\(\) Variances for E1/T1 CAS/R2”](#), on page 87.**gc\_GetResourceH()**

Supported.

**gc\_GetSigInfo()**

Not supported.

**gc\_GetUserInfo()**

Not supported.

**gc\_GetUsrAttr()**

Supported.

**gc\_GetVer()**

For Springware boards: Supported. For DM3 boards: Not supported.

**gc\_GetVoiceH()** (deprecated)

Supported.

**gc\_GetXmitSlot()**

For Springware boards: Supported (PDKRT only). For DM3 boards: Supported.

**gc\_HoldACK()**

Not supported.

**gc\_HoldCall()**Supported with variances described in [Section 8.2.12, “gc\\_HoldCall\(\) Variances for E1/T1 CAS/R2”](#), on page 87.**gc\_HoldRej()**

Not supported.

**gc\_InitXfer()**

Not supported.

**gc\_InvokeXfer()**

Not supported.

**gc\_LinedevToCCLIBID()**

Supported.

**gc\_Listen()**

For Springware boards: Supported (PDKRT only). For DM3 boards: Supported.

**gc\_LoadDxParm()**

For Springware boards: Supported (PDKRT only). For DM3 boards: Not supported.

**gc\_MakeCall()**

Supported with variances described in [Section 8.2.13, “gc\\_MakeCall\( \) Variances for E1/T1 CAS/R2”](#), on page 88.

**gc\_Open()** (deprecated)

Supported.

**gc\_OpenEx()**

Supported with variances described in [Section 8.2.14, “gc\\_OpenEx\( \) Variances for E1/T1 CAS/R2”](#), on page 89.

**gc\_QueryConfigData()**

For Springware boards: Supported (PDKRT only). For DM3 boards: Not supported.

**gc\_RejectInitXfer()**

Not supported.

**gc\_RejectModifyCall()**

Not supported.

**gc\_RejectXfer()**

Not supported.

**gc\_ReleaseCall()** (deprecated)

Supported.

**gc\_ReleaseCallEx()**

Supported.

**gc\_ReqANI()**

Not supported.

**gc\_ReqModifyCall()**

Not supported.

**gc\_ReqMoreInfo()**

For Springware boards: Supported (PDKRT only). For DM3 boards: Not supported.

**gc\_ReqService()**

Not supported.

**gc\_ResetLineDev()**

For Springware boards: Supported with variances described in [Section 8.2.15, “gc\\_ResetLineDev\( \) Variances for E1/T1 CAS/R2”](#), on page 91. For DM3 boards: Supported.

**gc\_RespService()**

Not supported.

**gc\_ResultInfo()**

Supported.

**gc\_ResultMsg()** (deprecated)

Supported.

**gc\_ResultValue()** (deprecated)

Supported.

**gc\_RetrieveAck()**

Not supported.

**gc\_RetrieveCall()**Supported with variances described in [Section 8.2.16, “gc\\_RetrieveCall\(\) Variances for E1/T1 CAS/R2”](#), on page 92.**gc\_RetrieveRej()**

Not supported.

**gc\_SendMoreInfo()**

For Springware boards: Supported (PDKRT only). For DM3 boards: Not supported.

**gc\_SetAlarmConfiguration()**

Supported.

**gc\_SetAlarmFlow()**

Supported.

**gc\_SetAlarmNotifyAll()**

Supported.

**gc\_SetAlarmParm()**

For Springware boards: Supported. For DM3 boards: Not supported.

**gc\_SetAuthenticationInfo()**

Not supported.

**gc\_SetBilling()**For Springware boards: Supported with variances described in [Section 8.2.17, “gc\\_SetBilling\(\) Variances for E1/T1 CAS/R2”](#), on page 92. For DM3 boards: Not supported.**gc\_SetCallingNum()** (deprecated)

Supported.

**gc\_SetCallProgressParm()**

For Springware boards: Supported (PDKRT only). For DM3 boards: Not supported.

**gc\_SetChanState()**Supported with variances described in [Section 8.2.18, “gc\\_SetChanState\(\) Variances for E1/T1 CAS/R2”](#), on page 92.**gc\_SetConfigData()**

For Springware boards: Supported (PDKRT only). For DM3 boards: Supported.

**gc\_SetEvtMsk()** (deprecated)For Springware boards: Supported with variances described in [Section 8.2.19, “gc\\_SetEvtMsk\(\) Variances for E1/T1 CAS/R2”](#), on page 93. For DM3 boards: Supported.



**gc\_SetInfoElem()**

Not supported.

**gc\_SetParm()**

Supported with variances described in [Section 8.2.20, “gc\\_SetParm\(\) Variances for E1/T1 CAS/R2”](#), on page 93.

**gc\_SetupTransfer()**

Supported with variances described [Section 8.2.21, “gc\\_SetupTransfer\(\) Variances for E1/T1 CAS/R2”](#), on page 94.

**gc\_SetUserInfo()**

Not supported.

**gc\_SetUsrAttr()**

Supported.

**gc\_SipAck()**

Not supported.

**gc\_SndFrame()**

Not supported.

**gc\_SndMsg()**

Not supported.

**gc\_Start()**

For Springware boards: Supported with variances described in [Section 8.2.22, “gc\\_Start\(\) and gc\\_Stop\(\) Variances for E1/T1 CAS/R2”](#), on page 94. For DM3 boards: Supported.

**gc\_StartTrace()**

For Springware boards: Supported with variances described in [Section 8.2.23, “gc\\_StartTrace\(\) Variances for E1/T1 CAS/R2”](#), on page 94. For DM3 boards: Not supported.

**gc\_Stop()**

For Springware boards: Supported with variances described in [Section 8.2.22, “gc\\_Start\(\) and gc\\_Stop\(\) Variances for E1/T1 CAS/R2”](#), on page 94. For DM3 boards: Supported.

**gc\_StopTrace()**

For Springware boards: Supported (PDKRT only). For DM3 boards: Not supported.

**gc\_StopTransmitAlarms()**

Supported.

**gc\_SwapHold()**

Supported with variances described in [Section 8.2.24, “gc\\_SwapHold\(\) Variances for E1/T1 CAS/R2”](#), on page 94.

**gc\_TransmitAlarms()**

Supported.

**gc\_UnListen()**

For Springware boards: Supported (PDKRT only). For DM3 boards: Supported

**gc\_util\_copy\_parm\_blk()**

Supported.

**gc\_util\_delete\_parm\_blk()**

Supported.

**gc\_util\_find\_parm( )**  
Supported.

**gc\_util\_find\_parm\_ex( )**  
Supported.

**gc\_util\_insert\_parm\_ref( )**  
Supported.

**gc\_util\_insert\_parm\_ref\_ex( )**  
Supported.

**gc\_util\_insert\_parm\_val( )**  
Supported.

**gc\_util\_next\_parm( )**  
Supported.

**gc\_util\_next\_parm\_ex( )**  
Supported.

**gc\_WaitCall( )**  
Supported.

## 8.2 Global Call Function Variances for E1/T1 CAS/R2

The Global Call function variances that apply when using E1/T1 CAS/R2 technology are described in the following sections. See the *Global Call API Library Reference* for generic (technology-independent) descriptions of the Global Call API functions.

- [gc\\_AcceptCall\( \) Variances for E1/T1 CAS/R2](#)
- [gc\\_AnswerCall\( \) Variances for E1/T1 CAS/R2](#)
- [gc\\_BlindTransfer\( \) Variances for E1/T1 CAS/R2](#)
- [gc\\_CallAck\( \) Variances for E1/T1 CAS/R2](#)
- [gc\\_Close\( \) Variances for E1/T1 CAS/R2](#)
- [gc\\_CompleteTransfer\( \) Variances for E1/T1 CAS/R2](#)
- [gc\\_Detach\( \) Variances for E1/T1 CAS/R2](#)
- [gc\\_DropCall\( \) Variances for E1/T1 CAS/R2](#)
- [gc\\_Extension\( \) Variances for E1/T1 CAS/R2](#)
- [gc\\_GetCallInfo\( \) Variances for E1/T1 CAS/R2](#)
- [gc\\_GetParm\( \) Variances for E1/T1 CAS/R2](#)
- [gc\\_HoldCall\( \) Variances for E1/T1 CAS/R2](#)
- [gc\\_MakeCall\( \) Variances for E1/T1 CAS/R2](#)
- [gc\\_OpenEx\( \) Variances for E1/T1 CAS/R2](#)
- [gc\\_ResetLineDev\( \) Variances for E1/T1 CAS/R2](#)
- [gc\\_RetrieveCall\( \) Variances for E1/T1 CAS/R2](#)
- [gc\\_SetBilling\( \) Variances for E1/T1 CAS/R2](#)

- [gc\\_SetChanState\( \)](#) Variances for E1/T1 CAS/R2
- [gc\\_SetEvtMsk\( \)](#) Variances for E1/T1 CAS/R2
- [gc\\_SetParm\( \)](#) Variances for E1/T1 CAS/R2
- [gc\\_SetupTransfer\( \)](#) Variances for E1/T1 CAS/R2
- [gc\\_Start\( \)](#) and [gc\\_Stop\( \)](#) Variances for E1/T1 CAS/R2
- [gc\\_StartTrace\( \)](#) Variances for E1/T1 CAS/R2
- [gc\\_SwapHold\( \)](#) Variances for E1/T1 CAS/R2

### 8.2.1 [gc\\_AcceptCall\( \)](#) Variances for E1/T1 CAS/R2

The **gc\_AcceptCall( )** function optionally responds to an inbound call request by providing an indication to the remote end that a call was received but not yet answered. This function causes ringback to be generated.

The **gc\_AcceptCall( )** function uses the **rings** parameter to specify the number of rings to wait before terminating the function, that is, before the Global Call API sends the GCEV\_ACCEPT event to the application.

- For PDK protocols, if the **rings** parameter is set to 0, the value of the **PSL\_AcceptCallDefaultNumOfRings** parameter in the country dependent parameters (.cdp) file is used.
- For ICAP protocols (Springware only), if the **rings** parameter is set to 0, the value specified in parameter \$9 of the country dependent parameters (.cdp) file is used.

### 8.2.2 [gc\\_AnswerCall\( \)](#) Variances for E1/T1 CAS/R2

The **gc\_AnswerCall( )** function indicates to the remote end that the connection is established (call has been answered). The **rings** parameter specifies the number of rings to wait before terminating the **gc\_AnswerCall( )** function; that is, before answering the call.

- For PDK protocols, if the **rings** parameter is set to 0, the value of the **PSL\_AnswerCallDefaultNumOfRings** parameter in the country dependent parameters (.cdp) file is used.
- For ICAP protocols (Springware only), if the **rings** parameter is set to 0, the value specified in parameter \$9 of the country dependent parameters (.cdp) file is used.

### 8.2.3 [gc\\_BlindTransfer\( \)](#) Variances for E1/T1 CAS/R2

The **gc\_BlindTransfer( )** function is only valid for applications using PDK protocols that support call hold and transfer. Check the **sys\_features** parameter in the .cdp file for a value of **Feature\_Transfer**. See the *Global Call Country Dependent Parameters (CDP) for PDK Protocols Configuration Guide* for more information.

## 8.2.4 **gc\_CallAck( ) Variances for E1/T1 CAS/R2**

**Note:** The variances described in this section apply when using Springware boards only. The **gc\_CallAck( )** function is not supported when using DM3 boards.

The **gc\_CallAck( )** function may be called before issuing a **gc\_AcceptCall( )** or a **gc\_AnswerCall( )** function to indicate to the network if more information is desired before completing the call. This function is used to request additional DDI digits from the network. After using this function, call the **gc\_GetCallInfo( )** function to retrieve the digits. The **gc\_GetCallInfo( )** function will return all DDI digits collected from the network (including both the digits already received and those returned by the network in response to the **gc\_CallAck( )** function call). Using the **gc\_CallAck( )** function for this service is described in the *Global Call API Library Reference*.

The valid range of values for the **gc\_CallAck( )** function **info\_len** field is from 1 to GCDG\_MAXDIGIT. If more than GCDG\_MAXDIGIT digits are required, or if an unknown number of digits is to be requested, set the **info\_len** field to GCDG\_NDIGIT.

The value GCDG\_PARTIAL may be ORed with the number of digits field if the application needs to call the **gc\_CallAck( )** function again for this call (that is, if the application needs additional DDI digits before accepting or rejecting the call).

## 8.2.5 **gc\_Close( ) Variances for E1/T1 CAS/R2**

The **gc\_Close( )** function only affects the link between the calling process and the device. For CAS protocols, if a voice resource is currently assigned to the specified line device, the voice resource will be closed. To keep the voice resource open for other operations, use the **gc\_Detach( )** function to detach the voice resource from the line device before issuing the **gc\_Close( )** function.

Functionality of **gc\_Close( )** is different for Springware and DM3 boards with regards to stopping the protocol.

### **Springware-specific variances**

Springware boards stop the protocol after **gc\_Close( )**.

### **DM3-specific variances**

DM3 boards do not stop the protocol after **gc\_Close( )**.

On DM3 boards, **gc\_Close( )** typically sets the protocol out of service; the protocol is not stopped until the board is stopped. Therefore, when a DM3 board uses a protocol that includes the **CDP\_ProtocolStopsOffhook** parameter, which determines the state of the hook switch signaling (on-hook or off-hook) when the protocol stops after **gc\_Close( )**, this parameter has no effect.

## 8.2.6 **gc\_CompleteTransfer( ) Variances for E1/T1 CAS/R2**

The **gc\_CompleteTransfer( )** function is only valid for applications using PDK protocols that support call hold and transfer. Check the **sys\_features** parameter in the .cdp file for a value of **Feature\_Transfer**. See the *Global Call Country Dependent Parameters (CDP) for PDK Protocols Configuration Guide* for more information.

## 8.2.7 **gc\_Detach( ) Variances for E1/T1 CAS/R2**

The **gc\_Detach( )** function logically disconnects a voice channel from a line device. It is the responsibility of the application to make sure that there is a voice resource available while the **gc\_WaitCall( )** function is active and that the current Global Call call state is Null or Idle. Furthermore, the **gc\_Detach( )** function can only be called in the Null, Idle, or Connected states.

## 8.2.8 **gc\_DropCall( ) Variances for E1/T1 CAS/R2**

The **gc\_DropCall( )** function supports the following values for its **cause** parameter:

**GC\_CALL\_REJECTED**

Call is not accepted

**GC\_NETWORK\_CONGESTION**

Cannot establish connection due to volume of traffic on network

**GC\_NORMAL\_CLEARING**

Normal end of call

**GC\_SEND\_SIT**

Sends a special information tone

**GC\_UNASSIGNED\_NUMBER**

Invalid called party number

**GC\_USER\_BUSY**

Called party is busy

**Note:** You must use the **dx\_stopch( )** function to terminate any application-initiated voice functions, such as **dx\_play( )** or **dx\_record( )**, before calling **gc\_DropCall( )**.

Some protocols do not support all **gc\_DropCall( )** causes for dropping a call. Any unsupported cause(s) is automatically mapped to the most appropriate cause. This approach facilitates developing protocol independent applications.

From the Accepted state, some protocols do not support a forced release of the line; that is, issuing a **gc\_DropCall( )** function after a **gc\_AcceptCall( )** function. Refer to the Protocol Limitations section in the *Global Call Country Dependent Parameters (CDP) for PDK Protocols Configuration Guide* for your protocol. If a forced release is attempted, the function fails and an error is returned. To recover, the application should issue a **gc\_AnswerCall( )** function followed by **gc\_DropCall( )** and **gc\_ReleaseCall( )** functions. However, anytime a **GCEV\_DISCONNECTED** event is received in the Accepted state, the **gc\_DropCall( )** function can be issued.

After the **gc\_AnswerCall()** function is issued, the application must wait for a GCEV\_ANSWER event. Otherwise the **gc\_DropCall()** function is ignored, no error is returned, and no drop call action is taken.

When using ICAPI protocols (Springware only), the **gc\_DropCall()** function occasionally results in the generation of the GCEV\_DROP\_CALL event followed by a GCEV\_BLOCKED event. The generation of the GCEV\_BLOCKED event is most likely if the **gc\_DropCall()** function is issued before the call is connected. The reason for the GCEV\_BLOCKED event is that the remote side does not recognize the disconnection in a timely manner. When the GCEV\_BLOCKED event occurs, call-related Global Call functions should not be issued until a GCEV\_UNBLOCKED event is detected on the respective device.

In some protocols, a **gc\_DropCall()** command on a call in the Accepted state requires a momentary transition to the Connected state. This may result in a charge being registered for the call.

## 8.2.9 **gc\_Extension()** Variances for E1/T1 CAS/R2

### DM3-specific variances

The **gc\_Extension()** function can be used to access the functionality of the Direct Signaling protocol. The Direct Signaling protocol is not a call control protocol; it is used strictly to give applications direct control over the signaling patterns on a line, as a means to allow the application to implement its own protocols. The Direct Signaling protocol allows the application to generate and detect signaling patterns. Applications can call the **gc\_Extension()** function to generate up to 11 distinct CAS patterns, and through the GCEV\_EXTENSION event, be notified when one of the patterns is detected by the protocol. For details about the Direct Signaling protocol and the **gc\_Extension()** function, see the *Global Call Country Dependent Parameters (CDP) for PDK Protocols Configuration Guide*.

## 8.2.10 **gc\_GetCallInfo()** Variances for E1/T1 CAS/R2

For E1 CAS and T1 robbed bit protocols that support enhanced call analysis (call progress), the **gc\_GetCallInfo()** **info\_id** CONNECT\_TYPE parameter contains the type of connection as returned by the function. These connection types are:

- GCCT\_CAD  
Connection due to cadence break
- GCCT\_PVD  
Connection due to voice detection
- GCCT\_PAMD  
Connection due to answering machine detection
- GCCT\_FAX  
Connection due to fax machine detection
- GCCT\_NA  
Connection type is not available

PDK protocols provide support for enhanced call analysis.

For protocols that do not support enhanced call progress analysis, the **gc\_GetCallInfo()** function with the **CONNECT\_TYPE** parameter specified will return a **CONNECT\_TYPE** value of **GCCT\_NA** (not available).

For E1 CAS protocols that support the **CALLINFOTYPE info\_id** parameter, a call information string containing either a **CHARGE** or **NO CHARGE** value is returned by the parameter; check your protocol in the *Global Call Country Dependent Parameters (CDP) for PDK Protocols Configuration Guide* for applicability.

## 8.2.11 **gc\_GetParm( )** Variances for E1/T1 CAS/R2

The **gc\_GetParm()** function retrieves the value of the specified parameter for a line device.

### Springware-specific variances

In addition to the **GCPR\_CALLINGPARTY** parameter, which is common across all technologies and documented in the *Global Call API Library Reference*, the following parameters are supported:

- **GCPR\_LOADTONES**
- **GCPR\_MEDIADetect**

See [Section 8.2.20, “gc\\_SetParm\( \) Variances for E1/T1 CAS/R2”](#), on page 93 for more information on the meaning of these parameters.

### DM3-specific variances

In addition to the **GCPR\_CALLINGPARTY** parameter, which is common across all technologies and documented in the *Global Call API Library Reference*, the following parameters are supported:

- **GCPR\_CALLPROGRESS**
- **GCPR\_MEDIADetect**
- **GCPR\_MINDIGITS**

See [Section 8.2.20, “gc\\_SetParm\( \) Variances for E1/T1 CAS/R2”](#), on page 93 for more information on the meaning of these parameters.

## 8.2.12 **gc\_HoldCall( )** Variances for E1/T1 CAS/R2

The **gc\_HoldCall()** function is only valid for applications using PDK protocols that support call hold and transfer. Check the **sys\_features** parameter in the .cdp file for a value of **Feature\_Hold**. See the *Global Call Country Dependent Parameters (CDP) for PDK Protocols Configuration Guide* for more information.

## 8.2.13 gc\_MakeCall( ) Variances for E1/T1 CAS/R2

gc\_MakeCall( ) function variances for E1/T1 CAS/R2 are discussed in the following topics:

- [Use of the timeout Parameter](#)
- [Other gc\\_MakeCall\( \) Considerations](#)

### 8.2.13.1 Use of the timeout Parameter

When using E1 CAS or T1 robbed bit line devices, the **timeout** parameter in the **gc\_MakeCall( )** function is supported when the **mode** parameter is set to either EV\_SYNC or EV\_ASYNC.

For ICAPI protocols (Springware only), when the **mode** parameter is set to EV\_ASYNC, the **timeout** parameter overrides the time-out parameter (\$13) value and the outbound number of ringback tones parameter (\$1) in most protocol country dependent parameters (.cdp) files.

- If the **timeout** parameter is set to 0, then the time-out and ringback parameters in the .cdp file are used to set the time-out conditions.
- If the **timeout** parameter is set to a value larger than a protocol time-out value, a protocol time-out may occur first, which will cause the **gc\_MakeCall( )** function to fail. The protocol time-out is configured in the .cdp file.
- If the **timeout** value is reached before the remote end answers the call, the application is notified of this condition and should respond as described in the **gc\_MakeCall( )** function description in the *Global Call API Library Reference*.
- If all **timeout** values are set to 0, no time-out condition will apply.

For PDK protocols, the time-out value used is determined by:

- The **timeout** parameter in the **gc\_MakeCall( )** function.
- The **PSL\_DefaultMakeCallTimeout** parameter specified in the .cdp file if the **timeout** parameter in the **gc\_MakeCall( )** function is 0 and call analysis is not specified.
- The **PSL\_CallProgressMaxDialingTime** parameter specified in the .cdp file if the **timeout** parameter in the **gc\_MakeCall( )** function is 0, call analysis is specified, and **PSL\_DefaultMakeCallTimeout** is less than **PSL\_CallProgressMaxDialingTime**.

**Note:** PDK protocols do not use the outbound number of ringback tones to define the time-out.

### 8.2.13.2 Other gc\_MakeCall( ) Considerations

If your T1 robbed bit circuit is provisioned for Feature Group A, your application should call the **gc\_MakeCall( )** function with a null dial string.

When using R2 protocols, a “#” in the dial string is not supported.

If a protocol error occurs during dialing and the default call progress enabled governs, then an error code or an event is returned as described in the **gc\_MakeCall( )** function description in the *Global Call API Library Reference*. If call progress is disabled and a protocol error occurs during dialing, then a GCRV\_BUSY result value or an EGC\_BUSY error is returned.



For drop and insert applications, call progress is typically disabled to enable the application to complete the dialing sequence, listen for voice or ringback on the line, and:

- if ringback is detected, to transition to the Alerting state
- if voice is detected, to transition to the Connected (call answered) state and to implement voice cut-through immediately

This methodology enables the application to pass signaling from the remote end (outbound line) to the caller on the inbound line. If call progress is not disabled, then the GCEV\_ALERTING event and the GCEV\_ANSWERED event may be received from the outbound line for an unacceptable amount of time after the dialing sequence was completed. During this period of time, the caller could misinterpret the silence on the line as a disconnect or a failure, and then hang up and redial. For further information, see the tips for programming drop and insert applications in the *Global Call API Programming Guide*.

In an E1 environment, the GCEV\_ALERTING event is generated when the equivalent of ringback is recognized. For almost all E1 protocols, this is a required part of the protocol, so E1 applications will receive the GCEV\_ALERTING event by default.

In a T1 environment, the GCEV\_ALERTING event is generated when the ringback is recognized. However, not all inbound applications will generate a ringback tone; for example, the PDK US MF protocol has disabled ringback tone generation by default to minimize call setup time. (Detecting the ringback tone takes several tenths of a second.) If the outbound application does not wish to use the detection of the ringback tone to generate the GCEV\_ALERTING event, the **CDP\_OUT\_Send\_Alerting\_After\_Dialing** parameter in the *pdsk\_us\_mf\_io.cdp* file should be set to 1 (default is 0). That way, if call progress is enabled, GCEV\_ALERTING is sent after dialing is initiated rather than when ringback is detected.

Since GCEV\_ALERTING is an optional event triggered by the inbound side, all applications must be able to handle not receiving the GCEV\_ALERTING event.

When the **gc\_MakeCall()** function sets up a call, the default is to enable call analysis (call progress). To change the enabled call progress default when making a call on Springware boards, see the **IC\_MAKECALL\_BLK** data structure (for ICAPI protocols) or the **PDK\_MAKECALL\_BLK** data structure (for PDK protocols) in [Chapter 9, “E1/T1 CAS/R2-Specific Data Structures”](#). (These structures do not apply to DM3 boards, which use **gc\_SetParm()** parameters to change call progress configuration as discussed in [Section 4.1.1, “Call Analysis with DM3 Boards”](#), on page 26.)

When the **gc\_MakeCall()** function sets up a call, the default is to enable call analysis (call progress). To change the enabled call progress default when making a call on Springware boards, use **PDK\_MAKECALL\_BLK** for PDK protocols and **IC\_MAKECALL\_BLK** for ICAPI protocols as discussed in [Chapter 9, “E1/T1 CAS/R2-Specific Data Structures”](#). (These structures do not apply to DM3 boards, which use **gc\_SetParm()** parameters to change call progress configuration as discussed in [Section 4.1.1, “Call Analysis with DM3 Boards”](#), on page 26.)

## 8.2.14 **gc\_OpenEx()** Variances for E1/T1 CAS/R2

The **gc\_OpenEx()** function is used to open both network board and channel (time slot) devices. This generic call control function initializes the specified time slot on the specified trunk. A line

device ID will be returned to the application. The E1 or T1 feature of this function specifies the voice device as part of the **devicename** parameter.

**gc\_OpenEx( )** function variances for E1/T1 CAS/R2 are discussed in the following topics:

- [Conventions for Specifying the devicename Parameter](#)
- [Other gc\\_OpenEx\( \) Considerations](#)
- [Handling GCEV\\_BLOCKED and GCEV\\_UNBLOCKED Events](#)

### 8.2.14.1 Conventions for Specifying the devicename Parameter

A device is specified by the **devicename** parameter using a format that includes protocol specific information.

The format for the fields used to specify this parameter is:

```
:N_<network_device_name>:P_<protocol_name>:V_<voice_channel_name>
```

The prefixes (N\_, P\_, and V\_) are used for parsing purposes. These fields may appear in any order. The fields within the **devicename** parameter must each begin with a colon.

The conventions described below allow the Global Call API to map subsequent calls made on specific line devices or CRNs to interface-specific libraries.

**<network\_device\_name>**

This field is required. It may be a board name or a time slot name:

- If <network\_device\_name> is a board name, use the format: dtiB<number of board>.
- If <network\_device\_name> is a time slot name, use the format: dtiB<number of board>T<number of time slot>.

**<protocol\_name>**

This field is required on Springware boards. It specifies the protocol to use. Use the root file name of the country dependent parameters (.cdp) file.

On DM3 boards, the protocol is determined at board initialization time and not when a Global Call device is opened. For compatibility, the <protocol\_name> field may be specified, but it is not used.

**<voice\_channel\_name>**

This field is optional depending on your application (see [Section 4.5, “Resource Allocation and Routing”](#), on page 45). It specifies the name of the voice channel to be associated with the device being opened. Use the following format: dxxxB<virtual board number>C<channel number>.

**Note:** Attachment to different types of DM3 voice devices is dependent on the protocol downloaded. For example, if one board has ISDN for protocols and another has T1 CAS, the T1 CAS network devices cannot be attached to the voice devices on the ISDN board. See the *Global Call API Programming Guide* for further information.

### 8.2.14.2 Other gc\_OpenEx( ) Considerations

For E1 CAS or T1 robbed bit applications, always specify a network resource (board or time slot level) and a protocol. A voice resource (<voice\_channel\_name>) may also be specified for E1 CAS or T1 robbed bit operations. When a voice resource is specified, Global Call automatically opens the voice device and internally attaches the voice device to the line device.

When using the CT Bus and a voice resource is specified, the **gc\_OpenEx( )** function routes the voice and network resources together.

When the voice resource is not specified, the application must perform these functions (open device, route, attach); see [Section 4.5, “Resource Allocation and Routing”](#), on page 45 for details.

When a network resource is specified, the **gc\_OpenEx( )** function internally issues a **dt\_open( )** function. Likewise, when a voice resource is specified, the **gc\_OpenEx( )** function internally issues a **dx\_open( )** function. The corresponding network or voice device handle may be retrieved using the **gc\_GetResourceH( )** function. These lower level device handles may be useful for routing or for playing or recording a file.

If a **gc\_OpenEx( )** function fails with an error value of EGC\_DXOPEN, then the internally issued **dx\_open( )** function failed. If a **gc\_OpenEx( )** function fails with an error value of EGC\_DTOPEN, then the internally issued **dt\_open( )** function failed.

### 8.2.14.3 Handling GCEV\_BLOCKED and GCEV\_UNBLOCKED Events

At the firmware level, when using Springware boards, the line is considered **unblocked** until otherwise informed (that is, some event occurs to change the state). From the Global Call perspective, the line is considered **blocked** until otherwise informed. To reconcile this difference in behavior, the Global Call software generates the required GCEV\_UNBLOCKED event as part of the **gc\_OpenEx( )** functionality with Springware boards.

When using Springware boards, if a blocking alarm exists on the line when an application tries to open a device, the **gc\_OpenEx( )** function will complete, generating the GCEV\_UNBLOCKED event, before the firmware detects that the alarm exists, which would trigger the generation of a GCEV\_BLOCKED event. This means that the application temporarily sees a GCEV\_UNBLOCKED event even though an alarm exists on the line. The application must be capable of handling a GCEV\_BLOCKED event at any time, even milliseconds after a GCEV\_UNBLOCKED event.

## 8.2.15 gc\_ResetLineDev( ) Variances for E1/T1 CAS/R2

### Springware-specific variances

For applications that use PDK protocols, the **gc\_ResetLineDev( )** function cannot be called while there is an alarm on the line.

For applications that use ICAPI protocols, the **gc\_ResetLineDev( )** function is not supported in synchronous mode. If the application calls **gc\_ResetLineDev( )** on a line device in synchronous mode (that is, with the mode parameter set to EV\_SYNC), the function fails silently.

### DM3-specific variances

There are no restrictions on using `gc_ResetLineDev()` with DM3 boards.

## 8.2.16 `gc_RetrieveCall()` Variances for E1/T1 CAS/R2

The `gc_RetrieveCall()` function is only valid for applications using PDK protocols that support call hold and transfer. Check the `sys_features` parameter in the `.cdp` file for a value of `Feature_Hold`. See the *Global Call Country Dependent Parameters (CDP) for PDK Protocols Configuration Guide* for more information.

## 8.2.17 `gc_SetBilling()` Variances for E1/T1 CAS/R2

**Note:** The variances described in this section apply when using Springware boards only. The `gc_SetBilling()` function is not supported when using DM3 boards.

On Springware boards, the `gc_SetBilling()` function sets different billing rates on a per call basis. For example:

- To charge the call, use `gc_SetBilling(crn, GCR_CHARGE, NULL, EV_SYNC)`
- To select no-charge for the call, use `gc_SetBilling(crn, GCR_NOCHARGE, NULL, EV_SYNC)`

The `gc_SetBilling()` function is called after the `GCEV_OFFERED` event arrives and before issuing a `gc_AcceptCall()` or `gc_AnswerCall()` function.

Not all protocols support this feature; see the *Global Call Country Dependent Parameters (CDP) for PDK Protocols Configuration Guide* for protocol specific limitations.

The `mode` parameter must be set to `EV_SYNC`. Asynchronous mode (`EV_ASYNC`) is not supported for this function.

## 8.2.18 `gc_SetChanState()` Variances for E1/T1 CAS/R2

The `GCLS_INSERVICE` and `GCLS_OUT_OF_SERVICE` states are the only valid service states that can be used to set the state of a line in an E1 CAS or T1 robbed bit environment.

**Note:** When a channel is set to out-of-service state, not all protocols send the blocking pattern by default. For such protocols, a parameter in the `.cdp` file has to be set to the appropriate value so that the blocking pattern is sent when the channel is put out-of-service. Refer to the *Global Call Country Dependent Parameters (CDP) for PDK Protocols Configuration Guide* for more information.

## 8.2.19 gc\_SetEvtMsk( ) Variances for E1/T1 CAS/R2

### Springware-specific variances

On Springware boards using PDK protocols, all of the mask parameter values are supported. See the **gc\_SetEvtMsk( )** function reference page in the *Global Call API Library Reference* for more information.

On Springware boards using ICAPI protocols, the following mask parameter values are supported:

- GCMSK\_ALERTING
- GCMSK\_BLOCKED
- GCMSK\_UNBLOCKED

### DM3-specific variances

There are no restrictions on using **gc\_SetEvtMsk( )** with DM3 boards. All of the mask parameter values are supported. See the **gc\_SetEvtMsk( )** function reference page in the *Global Call API Library Reference* for more information.

## 8.2.20 gc\_SetParm( ) Variances for E1/T1 CAS/R2

The **gc\_SetParm( )** function sets the default parameters and all channel information associated with the specific line device. In addition to the GCPR\_CALLINGPARTY parameter, which is common across all technologies and documented in the *Global Call API Library Reference*, the parameters listed in Table 22 are supported.

**Table 22. Parameters Supported, gc\_GetParm( ) and gc\_SetParm( )**

Parameter	Level	Description	Supported on
GCPR_CALLPROGRESS	channel	Enables or disables call progress; enabled by default. If this parameter is disabled, post-connect call progress is also disabled, regardless of the setting of GCPR_MEDIADETECT.	DM3
GCPR_LOADTONES	channel	Enables or disables downloading of predefined call progress tones to the firmware. These tones are predefined in the E1 CAS or T1 robbed bit specific configuration files and are used for call progress. The tones are downloaded during execution of the <b>gc_Attach( )</b> or <b>gc_AttachResource( )</b> function.	Springware
GCPR_MEDIADETECT	channel	Enables or disables post-connect call progress or media detection; disabled by default.	DM3, Springware
GCPR_MINDIGITS	channel	Specifies the minimum number of digits to receive before a call is offered to the application.	DM3

For further information about the GCPR\_CALLPROGRESS, GCPR\_LOADTONES, and GCPR\_MEDIADetect parameters, see [Section 4.1, “Call Progress and Call Analysis”](#), on page 25.

## 8.2.21 **gc\_SetupTransfer( ) Variances for E1/T1 CAS/R2**

The **gc\_SetupTransfer( )** function is only valid for applications using PDK protocols that support call hold and transfer. Check the **sys\_features** parameter in the .cdp file for a value of **Feature\_Transfer**. See the *Global Call Country Dependent Parameters (CDP) for PDK Protocols Configuration Guide* for more information.

## 8.2.22 **gc\_Start( ) and gc\_Stop( ) Variances for E1/T1 CAS/R2**

### **Springware-specific variances**

On Springware boards using PDK protocols, the **gc\_Start( )** function is used to access the error and debug logging capabilities of the PDKRT call control library. See [Section 7.2, “Debugging Applications that Use PDK Protocols”](#), on page 67 for more information.

On Springware boards using ICAPI protocols, when the **gc\_Start( )** function is called, a log file, if enabled, is created. This file logs debug information for all ICAPI call control libraries for all open channels. The log file remains open until the **gc\_Stop( )** function is called. This allows channels to be opened, closed, and reopened multiple times without overwriting or otherwise affecting the continuity of the log file. See [Section 7.3, “Debugging Applications that Use ICAPI Protocols”](#), on page 72 for more information.

### **DM3-specific variances**

There are no restrictions on using **gc\_Start( )** and **gc\_Stop( )** with DM3 boards.

## 8.2.23 **gc\_StartTrace( ) Variances for E1/T1 CAS/R2**

**Note:** The variances described in this section apply when using Springware boards only. The **gc\_StartTrace( )** function is not supported when using DM3 boards.

When using PDK protocols, the **gc\_StartTrace( )** function can be used to enable logging on individual channels. This function has no effect unless the name of the log file and the logging level have been set using the **gc\_Start( )** function. The **gc\_StartTrace( ) filename** parameter is ignored. The name of the log file is specified in the PDK\_START\_STRUCT data structure. See [Section 7.2, “Debugging Applications that Use PDK Protocols”](#), on page 67 for more information.

## 8.2.24 **gc\_SwapHold( ) Variances for E1/T1 CAS/R2**

The **gc\_SwapHold( )** function is only valid for applications using PDK protocols that support call hold and transfer. Check the **sys\_features** parameter in the .cdp file for a value of **Feature\_Transfer**.



See the *Global Call Country Dependent Parameters (CDP) for PDK Protocols Configuration Guide* for more information.





# E1/T1 CAS/R2-Specific Data Structures

---

This chapter describes the data structures that are specific to E1/T1 CAS/R2 technology.

- [IC\\_MAKECALL\\_BLK..... 98](#)
- [PDK\\_MAKECALL\\_BLK..... 99](#)

**Note:** These data structures are used with Springware boards only.

## IC\_MAKECALL\_BLK

```
typedef struct ic_makecall_blk{
    unsigned long    flags;
    void             *v_rfu_ptr;
    unsigned long    ul_rfu[4];
}IC_MAKECALL_BLK;
```

### ■ Description

For Springware boards using ICAPI protocols, the IC\_MAKECALL\_BLK structure contains information used by the **gc\_MakeCall()** function when setting up a call. When the **gc\_MakeCall()** function sets up a call, the default is to enable call analysis (call progress). This default can be changed on a call basis by setting the **flags** parameter in the IC\_MAKECALL\_BLK data structure.

### ■ Field Descriptions

The fields of the IC\_MAKECALL\_BLK data structure are described as follows:

flags

Controls call analysis on a per call basis. The flags included are:

- NO\_CALL\_PROGRESS – Set to 0 to enable call analysis (default). Set to 1 to disable call analysis.

\*v\_rfu\_ptr

Reserved for future use.

ul\_rfu[4]

Reserved for future use.

## PDK\_MAKECALL\_BLK

```
typedef struct pdk_makecall_blk{
    unsigned long    flags;
    void             *v_rfu_ptr;
    unsigned long    ul_rfu[4];
}PDK_MAKECALL_BLK;
```

### ■ Description

For Springware boards using PDK protocols, the PDK\_MAKECALL\_BLK structure contains information used by the **gc\_MakeCall()** function when setting up a call. When the **gc\_MakeCall()** function sets up a call, the default is to enable call analysis (call progress). This default can be changed on a call basis by setting the **flags** parameter in the PDK\_MAKECALL\_BLK data structure.

**Note:** Control of call progress and media detection at **gc\_MakeCall()** time works only when the following parameters in the .cdp file are set to allow application control:

```
/* Set to 0 to disable, 1 to enable, and 2 to allow app control */
R4 INTEGER_t PSL_MakeCall_CallProgress = 0
DM3 INTEGER_t PSL_CACallProgressOverride = 0

/* Set 1 to enable, 2 to allow app control */
R4 INTEGER_t PSL_MakeCall_MediaDetect = 2
DM3 INTEGER_t PSL_CAMediaDetectOverride = 2
```

### ■ Field Descriptions

The fields of the PDK\_MAKECALL\_BLK data structure are described as follows:

**flags**

Contains a bitmask that controls call analysis and media type detection on a per call basis. The possible values that can be ORED are:

- NO\_CALL\_PROGRESS – To disable call analysis.
- MEDIA\_TYPE\_DETECT – To enable media type detection.

**\*v\_rfu\_ptr**

Reserved for future use.

**ul\_rfu[4]**

Reserved for future use.

### ■ Example

```
/* To enable Media Detection and disable CPA*/
if (disableCPA && enableMediaDetection)
{
    m_pdkMakecallBlk.flags |= (NO_CALL_PROGRESS|MEDIA_TYPE_DETECT);
    m_gcMakecallBlk.cclib = &m_pdkMakecallBlk;
}

/* To disable CPA */
if (disableCPA)
{
    m_pdkMakecallBlk.flags |= NO_CALL_PROGRESS;
    m_gcMakecallBlk.cclib = &m_pdkMakecallBlk;
}
```

```
/* To enable Media Detection */  
if (enableMediaDetection)  
{  
    m_pdkMakecallBlk.flags |= MEDIA_TYPE_DETECT;  
    m_gcMakecallBlk.cclib = &m_pdkMakecallBlk;  
}
```

# E1/T1 CAS/R2-Specific Event Cause Values

## 10

This chapter lists the supported E1/T1 CAS/R2-specific event cause values, which are retrieved by `gc_ResultValue()` and `gc_ResultInfo()`, and provides a description of each value.

**Note:** The information in this chapter is applicable to DM3 boards only.

Table 23 lists the E1/T1 CAS/R2 call control library cause values supported by DM3 boards.

**Table 23. Call Control Library Cause Values When Using DM3 Boards**

Cause Value (Decimal)	Cause Value (Hex)	Description
128	0x80	Requested information available. No more expected.
129	0x81	Requested information available. More expected.
130	0x82	Some of the requested information available. Timeout.
131	0x83	Some of the requested information available. No more expected.
132	0x84	Requested information not available. Timeout.
133	0x85	Requested information not available. No more expected.
134	0x86	Information has been sent successfully.
<b>Note:</b> The cause values in this table are ORed with the value 0x300, which identifies them as call control library cause values.		

Table 24 lists the firmware-related cause values supported by DM3 boards.

**Table 24. Firmware-Related Cause Values When Using DM3 Boards**

Cause Value (Decimal)	Cause Value (Hex)	Description
01	0x01	Busy
02	0x02	Call Completion
03	0x03	Canceled
04	0x04	Network congestion
05	0x05	Destination busy
06	0x06	Bad destination address
07	0x07	Destination out of order
08	0x08	Destination unreachable
<b>Note:</b> The cause values in this table are ORed with the value 0xC0, which identifies them as firmware-related cause values.		

Table 24. Firmware-Related Cause Values When Using DM3 Boards (Continued)

Cause Value (Decimal)	Cause Value (Hex)	Description
09	0x09	Forward
10	0x0A	Incompatible
11	0x0B	Incoming call
12	0x0C	New call
13	0x0D	No answer from user
14	0x0E	Normal clearing
15	0x0F	Network alarm
16	0x10	Pickup
17	0x11	Protocol error
18	0x12	Redirection
19	0x13	Remote termination
20	0x14	Call rejected
21	0x15	Special Information Tone (SIT)
22	0x16	SIT Custom Irregular
23	0x17	SIT No Circuit
24	0x18	SIT Reorder
25	0x19	Transfer
26	0x1A	Unavailable
27	0x1B	Unknown cause
28	0x1C	Unallocated number
29	0x1D	No route
30	0x1E	Number changed
31	0x1F	Destination out of order
32	0x20	Invalid format
33	0x21	Channel unavailable
34	0x22	Channel unacceptable
35	0x23	Channel not implemented
36	0x24	No channel
37	0x25	No response
38	0x26	Facility not subscribed
39	0x27	Facility not implemented
40	0x28	Service not implemented
41	0x29	Barred inbound
<b>Note:</b> The cause values in this table are ORed with the value 0xC0, which identifies them as firmware-related cause values.		

**Table 24. Firmware-Related Cause Values When Using DM3 Boards (Continued)**

Cause Value (Decimal)	Cause Value (Hex)	Description
42	0x2A	Barred outbound
43	0x2B	Destination incompatible
44	0x2C	Bearer capability unavailable
<b>Note:</b> The cause values in this table are ORed with the value 0xC0, which identifies them as firmware-related cause values.		





# Supplementary Reference Information

---

# 11

This chapter lists references to publications about E1/T1 CAS/R2 technology.

For additional information about E1 or T1 telephony, see the following publications:

- R2 MF Signaling References
  - *Specifications of Signaling Systems R1 and R2*, International Telegraph and Telephone Consultative Committee (CCITT), Blue Book Vol. VI, Fascicle VI.4, ISBN 92-61-03481-0
  - *General Recommendations on Telephone Switching and Signaling*, International Telegraph and Telephone Consultative Committee (CCITT), Blue Book Vol. VI, Fascicle VI.1, ISBN 92-61-03451-9
- T1 Robbed Bit Signaling References
  - Bellamy, John, *Digital Telephony*, 2nd ed. New York: John Wiley & Sons, 1991
  - Fike, John L., and George Friend, *Understanding Telephone Electronics*, Indiana: Howard W. Sams & Company, 1988
  - Flanagan, William A., *The Guide to T-1 Networking*, 4th ed. New York, Telecom Library Inc., 1990
  - *LATA Switching Systems Generic Requirements (LSSGR)*, Bellcore Technical Reference TR-TSY-000064, Issue 2, July 1987, and modules, Bellcore



## Symbols

@0

ICAPI special parameter 64

## A

address signals 17

alarm handling 49, 52

analog links 28

ANI 17

answering machine detection 32, 86

automatic number identification 17

## B

backward signal 17

billing rates 92

B-tones 17

## C

cadence break 32, 86

call analysis 14, 25

call progress 25

call progress tones 16, 36

called party 17

calling party 17

CAS pattern signal declarations 37

CAS\_SIGNAL\_PULSE\_t 38

CAS\_SIGNAL\_TRAIN\_t 39

CAS\_SIGNAL\_TRANS\_t 37

central office 17

CO 17

code example

call progress tones 36

compelled signaling 19

connect detection 36

CPE 17

customer premises equipment 17

## D

D4 frame 15

D4 superframe 15

DDI 20

DDI digits 84

debugging applications

ICAPI protocols 72

PDK protocols 67

dedicated voice resource

example of 46

destination CO 14

dial tone 13

dialed number identification service 20

DID 20

direct dialing in 20

direct inward dialing 20

direction indicator

in protocol name 62

dm3cc\_parm.h

header file 65

DNIS 20

DTMF 13

dynamic trunk configuration 40

## E

E&M

interface 15

signals 15

E1 protocol name 62

ESF 16

event cause values 101

extended superframe 16

## F

fax machine detection 32, 86

flash-hook 15

forward signal 17

frequency overlap 13

## G

gc\_AcceptCall( ) 83

gc\_AnswerCall( ) 83

gc\_Attach( ) 29, 45

- gc\_AttachResource(\_) 29
- gc\_BlindTransfer(\_) 83
- gc\_CallAck( ) 84
- gc\_Close(\_) 84
- gc\_CompleteTransfer( ) 85
- gc\_Detach(\_) 45, 85
- gc\_DropCall(\_) 85
- gc\_Extension(\_) 86
- gc\_GetCallInfo(\_) 26, 28, 32, 86
- gc\_GetConfigData(\_) 56, 57
- gc\_GetParm(\_) 87
- gc\_GetResourceH(\_) 45
- gc\_HoldCall( ) 87
- gc\_LoadDxParm(\_) 28
- gc\_MakeCall(\_) 26, 30, 88
- gc\_OpenEx(\_) 29, 45, 89
  - devicename parameter 90
- gc\_QueryConfigData(\_) 57
- gc\_ResetLineDev(\_) 91
- gc\_ResultInfo( ) 101
- gc\_ResultValue(\_) 26, 101
- gc\_RetrieveCall( ) 92
- gc\_SetBilling( ) 92
- gc\_SetChanState(\_) 92
- gc\_SetConfigData(\_) 56, 57
- gc\_SetEvtMsk(\_) 93
- gc\_SetParm(\_) 26, 29, 31, 93
- gc\_SetupTransfer( ) 94
- gc\_Start(\_) 94
- gc\_StartTrace( ) 94
- gc\_Stop(\_) 94
- gc\_SwapHold( ) 94
- gc\_WaitCall(\_) 85
- GCEV\_ALERTING event 89
- GCEV\_ANSWERED event 89
- GCEV\_BLOCKED event 49, 52
- GCEV\_UNBLOCKED event 49, 52
- gcpdkrt.h
  - header file 65
- GCPR\_CALLPROGRESS 26, 93
- GCPR\_LOADTONES 29, 93
- GCPR\_MEDIADTECT 26, 27, 31, 93
- GCPR\_MINDIGITS 93
- Group A backward signal 18
- Group B backward signal 18
- Group I forward signal 18
- Group II forward signal 18

GTD 35

## I

- IC\_MAKECALL\_BLK structure 98
- ICAPI protocol
  - debugging applications 72
  - file set 63
- icapi.cfg file 72
- icapi.h
  - header file 65
- icapi.inf file
  - generation of 73
- incoming register 17
  - backward signals 18
- international networks 18
- interregister signals 17

## L

- local CO 14
- local loop 13
- log file 67
  - gc\_Start(\_) 94
  - ICAPI protocols 72
- logging
  - enabling and disabling for PDK protocols 68
  - enabling for ICAPI protocols 73
  - number of channels to monitor 73

## M

- media type detection 25
- MF
  - description 13
- MF SOCOTEL
  - protocol name 62
  - signaling 19

## N

- national networks 18
- national traffic 17
- network resource 44

## O

- off-hook 15
- operator intercept 14
- outbound call 17



outgoing register 17

## P

PCM carrier system 16

PDK protocol

debugging applications 67

file set 63

PDK protocol parameters 57

PDK\_MAKECALL\_BLK structure 99

PDKRT call control library 56

pdktrace tool 67

protocol

dynamically configuring 43

file set for ICAPI 63

file set for PDK 63

sample component names 63

protocol component

.cdp file 64

protocol module 63

protocol module

ICAPI 63

PDK 63

protocol name

country code 62

direction indicator 62

E&M 62

E1 protocol 62

MF SOCOTEL 62

protocol type 62

R1 MFC protocol 62

R2 protocol 62

T1 E&M with MF protocol 62

protocol service layer parameters 57

protocol state information parameters 57

protocol version 59

pulse dialing 13

## R

R1 MFC protocol name 62

R2 MF

compelled signaling 19

forward signal 18

multifrequency combinations 18

signaling 16

signaling concepts 17

R2 MFC protocol name 62

R2 tones 16

rates

billing 92

resource association 44

resource sharing 44

ringback 14

ringback tone 28

ringing tone 14

robbed bit signaling 15

rotary dialing 13

## S

service states 92

setting up a call 98, 99

SF 15

shared voice resource

example 47

signaling bits 15

signaling concepts

R2 MF 17

single frequency 15

Socotel backbone 18

supervisory signaling

R2 MF 17

## T

T1 E&M protocol name 62

T1 trunk 15

tonal information

R2 17

tone dialing 13

tone template 35

commenting out 35

TONEOFF event 36

TONEON event 36

trunk configuration

dynamic 40

dynamically setting the protocol 43

## V

voice detection 32, 86

voice resource

attaching 45

dedicated 44, 45

detaching 45

shared 47

