

GlobalCall™ E-1/T-1 Technology User's Guide for Linux and Windows

Copyright © 2001 Dialogic Corporation

05-0615-008

COPYRIGHT NOTICE

All contents of this document are subject to change without notice and do not represent a commitment on the part of Dialogic Corporation. Every effort is made to ensure the accuracy of this information. However, due to ongoing product improvements and revisions, Dialogic Corporation cannot guarantee the accuracy of this material, nor can it accept responsibility for errors or omissions. No warranties of any nature are extended by the information contained in these copyrighted materials. Use or implementation of any one of the concepts, applications, or ideas described in this document or on Web pages maintained by Dialogic may infringe one or more patents or other intellectual property rights owned by third parties. Dialogic does not condone or encourage such infringement. Dialogic makes no warranty with respect to such infringement, nor does Dialogic waive any of its own intellectual property rights which may cover systems implementing one or more of the ideas contained herein. Procurement of appropriate intellectual property rights and licenses is solely the responsibility of the system implementer. The software referred to in this document is provided under a Software License Agreement. Refer to the Software License Agreement for complete details governing the use of the software.

All names, products, and services mentioned herein are the trademarks or registered trademarks of their respective organizations and are the sole property of their respective owners. DIALOGIC (including the Dialogic logo), DTI/124, and SpringBoard are registered trademarks of Dialogic Corporation. A detailed trademark listing can be found at: <http://www.dialogic.com/legal.htm>.

Publication Date: October, 2001

Part Number: 05-0615-008

Dialogic, an Intel Company
1515 Route 10
Parsippany NJ 07054
U.S.A.

For **Technical Support**, visit the Dialogic support website at:
<http://support.dialogic.com>

For **Sales Offices** and other contact information, visit the main Dialogic website at:
<http://www.dialogic.com>

Table of Contents

1. How to Use This Guide	1
1.1. Organization of this Guide.....	1
1.2. Dialogic Products That Support E-1/T-1 Signaling	2
2. Signaling Concepts	3
2.1. Making Telephone Calls: Transmission of Digits and Signaling Information.....	3
2.1.1. Making Long Distance and Global Telephone Calls	6
2.2. T-1 Robbed Bit Signaling Concepts	6
2.3. E-1 CAS Signaling Concepts.....	7
2.4. R2 MF Signaling Concepts.....	8
2.4.1. R2 MF Multifrequency Combinations	10
2.4.2. R2 MF Signal Meanings	10
2.4.3. R2 MF Compelled Signaling	12
2.5. Direct Dialing In (DDI) Service	13
3. Developing GlobalCall E-1 CAS or T-1 Robbed Bit Applications	15
3.1. GTD Tone Considerations	15
3.2. Call Analysis.....	16
3.2.1. Call Analysis Functionality for ICAPI Protocols	17
3.2.2. Call Analysis Functionality for PDK Protocols	20
3.3. Header Files.....	27
3.4. Resource Association.....	27
3.5. Alarm Handling	28
3.6. Run Time Configuration of the PDKRT Call Control Library	28
3.7. Run Time Configuration of PDK Protocol Parameters.....	29
3.8. Determining the Protocol Version	33
4. Applying GlobalCall Functions to E-1 CAS or T-1 Robbed Bit Applications	37
4.1. gc_AcceptCall().....	37
4.2. gc_AnswerCall().....	38
4.3. gc_CallAck()	38
4.4. gc_BlindTransfer()	39
4.5. gc_Close().....	39
4.6. gc_CompleteTransfer().....	39
4.7. gc_Detach()	39
4.8. gc_DropCall().....	39

GlobalCall™ E-1/T-1 Technology User's Guide for Linux and Windows

4.9. gc_Extension().....	41
4.10. Extension Event.....	42
4.11. gc_GetCallInfo().....	42
4.12. gc_GetCallProgressParm().....	43
4.13. gc_HoldCall().....	43
4.14. gc_MakeCall().....	43
4.14.1. IC_MAKECALL_BLK.....	45
4.14.2. PDK_MAKECALL_BLK.....	46
4.15. gc_OpenEx().....	47
4.15.1. Conventions for Specifying the devicename Parameter.....	47
4.15.2. Examples of the devicename Parameter.....	48
4.15.3. Other gcOpen() and gc_OpenEx() Considerations.....	48
4.16. gc_RetrieveCall().....	49
4.17. gc_SetBilling().....	49
4.18. gc_SetCallProgressParm().....	50
4.19. gc_SetChanState().....	50
4.20. gc_SetEvtMsk().....	50
4.21. gc_SetParm().....	51
4.22. gc_Start() and gc_Stop().....	51
4.23. gc_StartTrace().....	51
4.24. gc_SetupTransfer().....	52
4.25. gc_SwapHold().....	52
5. Resource Allocation and Routing.....	53
5.1. Dedicated Voice Resources.....	53
5.1.1. Dedicated Voice Resources Example.....	55
5.2. Shared Voice Resources.....	56
5.2.1. Shared Voice Resources - LINUX and Windows Example.....	57
6. Protocols.....	59
6.1. Protocols Supported.....	59
6.2. Protocol Naming Convention.....	60
6.3. Protocol Components.....	62
6.3.1. Country Dependent Parameter (.cdp) Files.....	63
6.3.2. PDK Site Dependent Parameter (.sdp) Files.....	64
6.3.3. Parameter (.prm) Files (LINUX only).....	65
6.3.4. Parameter (.prm) Files (Windows only).....	65
6.4. Using ICAPI and PDK Protocols.....	66
7. ICAPI Configuration Parameters.....	69
8. Debugging Applications.....	73

Table of Contents

8.1. Debugging Applications that use ICAPAPI Protocols	73
8.2. Debugging Applications that use PDK Protocols	74
8.2.1. Enabling and Disabling the Logging.....	74
Appendix A - Related Publications	83
R2 MF Signaling References	83
T-1 Robbed Bit Signaling References	83
Appendix B - Sample .sdp File.....	85
Index	89

GlobalCall™ E-1/T-1 Technology User's Guide for Linux and Windows

List of Tables

Table 1. Signaling Used to Dial (Hz)	4
Table 2. TONE_t Signal Definition Parameters	24
Table 3. Configurable PDKRT Call Control Library Parameters	29
Table 4. CDP Parameters.....	30
Table 5. PSL and SYS Parameters	31
Table 6. Configurable PDK Protocol Parameters	32
Table 7. IC_MAKECALL_BLK Field Descriptions	46
Table 8. PDK_MAKECALL_BLK Field Descriptions	46
Table 9. Protocol Naming Convention	60
Table 10. Protocol Component Name Structure	61
Table 11. Sample ICAPI Protocol File Set	62
Table 12. Sample PDK Protocol File Set	62
Table 13. <i>icapi.cfg</i> File Parameters	70
Table 14. <i>cclib_data</i> Fields and Values	75
Table 15. <i>log_level</i> Values	77
Table 16. <i>log_service</i> Values	79
Table 17. <i>log_cachedump</i> Values	80
Table 18. Sample <i>log_channel</i> Values	80

GlobalCall™ E-1/T-1 Technology User's Guide for Linux and Windows

1. How to Use This Guide

This guide is for users who use GlobalCall Application Programming Interface (API), the GlobalCall Interface Control API (ICAPI) call control library or the GlobalCall Protocol Development Kit Run Time (PDKRT) call control library and related software to develop LINUX or Windows applications in an E-1 environment using channel associated signaling (CAS) or in a T-1 robbed bit environment.

Use this guide in conjunction with the following manuals:

- *GlobalCall API Software Reference*
- *GlobalCall Application Developers Guide*
- *GlobalCall Country Dependent Parameters (CDP) Reference*

Products covered by this guide and the organization of this guide are described in this chapter. Differences between the implementation of a GlobalCall application in a LINUX or a Windows environment are either described parenthetically or are presented in separate paragraphs/sections.

Information that is specific to the use of the ICAPI or PDKRT call control library is identified explicitly.

1.1. Organization of this Guide

This guide provides information for developing E-1 CAS and T-1 robbed bit GlobalCall applications when using the GlobalCall ICAPI call control library and details differences in GlobalCall function usage in E-1 CAS or T-1 robbed bit applications as follows:

Chapter 2 provides an overview of E-1 CAS and T-1 robbed bit signaling concepts

Chapter 3 presents guidelines for developing E-1 CAS or T-1 robbed bit applications.

Chapter 4 describes the additional functionality of specific GlobalCall functions used for developing E-1 CAS or T-1 robbed bit applications.

Chapter 5 describes using dedicated or shared voice resources in an E-1 or a T-1 robbed bit environment.

Chapter 6 describes the protocol conventions used and programming considerations when incorporating individual country protocol(s) into your application.

Chapter 7 describes the diagnostic tools available for debugging a GlobalCall application.

Appendix A lists related publications for further information on E-1 or T-1 telephony.

An **Index** follows the appendix.

1.2. Dialogic Products That Support E-1/T-1 Signaling

The GlobalCall software provides a consistent interface across Dialogic products interfaced to various networks (for example, T-1 ISDN, E-1 ISDN, E-1 CAS and T-1 robbed bit). See the Release Catalog for your operating system for the Dialogic product combinations that provide CAS signaling and T-1 robbed bit signaling capabilities.

2. Signaling Concepts

This chapter provides an overview of how digits and signaling information is transmitted when a telephone call is made and describes the E-1 CAS, T-1 Robbed Bit, and R2 MF, and Direct Dialing In (DDI) signaling concepts.

2.1. 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. Signaling Used to Dial*). 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. Signaling Used to Dial*). 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 *DTMF and MF Tone Specifications Appendix* in the *Voice Software Reference - Programmer's Guide* for your operating system for details.

The MF digits are typically used for CO-to-CO signaling.

Table 1. Signaling Used to Dial (Hz)

Code	Pulse	DTMF	MF	R2 MF Forward	R2 MF 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,

2. Signaling Concepts

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 varies 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 3 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 or by dropping loop current briefly. 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 by dropping loop current. However, some telcos don't send a disconnect signal; therefore a local CO must use other methods to detect a remote disconnect.

2.1.1. 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, T-1 digital trunks, E-1 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.

2.2. T-1 Robbed Bit Signaling Concepts

A T-1 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 T-1 trunk. Twelve D4 frames make up a D4 superframe.

Signaling information is carried on a T-1 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 T-1 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

2. Signaling Concepts

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 4 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.

2.3. E-1 CAS Signaling Concepts

An E-1 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)

This method of using a separate time slot (channel 16) to carry signaling information is called Channel Associated Signaling (CAS). E-1 CAS service is available in Europe 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 E-1 areas will be used. In addition, the E-1 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 E-1 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. R2 MF 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 R2 MF signaling, call information can 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 tones 103 and 104 and ringback tone 105) may be useful in determining the condition of a call.

The following paragraphs describe R2 MF signaling as it is used in a full R2 network, global tone detection considerations and GlobalCall call analysis capability. See also *Section 6.3.1. Country Dependent Parameter (.cdp) Files* for using call progress tones (busy tones 103 and 104 and ringback tone 105) in determining the condition of a call. The protocols do not require call analysis because the called party condition is received via R2 tones; but you can modify the *.cdp* file so that the protocol can use either call progress tones or call analysis. GlobalCall always uses call progress tones and always detects a busy tone.

2.4. R2 MF Signaling Concepts

R2 MF signaling is an international signaling system that transmits numerical and other information relating to the called and the calling subscribers' lines. R2 MF 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

2. Signaling Concepts

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.

R2 MF 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 ICAP 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 T-1 span, the B-tones (also called Group B signals, see *Section 2.4.2. R2 MF Signal Meanings*) 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 busy 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.

2.4.1. R2 MF Multifrequency Combinations

R2 MF 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 R2 MF 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 Software Reference – Features Guide* for your operating system for lists of these signal tone pairs.

2.4.2. R2 MF 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.

2. Signaling Concepts

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 suppresser

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 Software Reference – Features Guide* for your operating system for more details and definitions of R2 MF signals.

2.4.3. R2 MF Compelled Signaling

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

- R2 MF 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 GlobalCall 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.

2. Signaling Concepts

R2 MF 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.

2.5. Direct Dialing In (DDI) Service

Since DTMF, MF and R2 MF 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.

GlobalCall™ E-1/T-1 Technology User's Guide for Linux and Windows

3. Developing GlobalCall E-1 CAS or T-1 Robbed Bit Applications

This chapter offers advice and suggestions for programmers designing and coding GlobalCall applications in a LINUX or Windows environment. Specific guidelines for developing E-1 CAS or T-1 robbed bit applications when using the GlobalCall ICAPI call control library are provided. Topics include the following:

- GTD tone considerations
- GlobalCall call analysis capability
- header files
- resource association
- alarm handling

3.1. GTD Tone Considerations

Unless Global Tone Detection (GTD) is disabled by the application by calling the **gc_SetParm()** function, if a protocol uses GTD tones for call analysis, the protocol sends tone definitions to the firmware when the **gc_Attach()** function is issued. The **gc_Attach()** function may be called directly from the application; this function is also called from within the **gc_OpenEx()** function if a voice channel is specified. The voice channel must be idle. Any pre-existing tones are deleted.

CAUTION

The application must NOT delete the default tones downloaded by the protocol or the protocol will fail.

If the application requires additional tones after the initial set of tones are loaded, they must be redefined after calling the **gc_Attach()** function. For any application that calls the **gc_Attach()** function several times on the same device (for example, when using resource sharing), the overhead associated with redundant tone deletion and definition may be avoided by calling the **gc_SetParm(idev,**

GCPR_LOADTONES, GCPV_DISABLE) function after the first call to the **gc_Attach()** function. Afterward, the application must reverse the effect of the **gc_SetParm()** function by issuing a **gc_SetParm(ldev, GCPR_LOADTONES, GCPV_ENABLE)** function when the call is complete.

NOTE: For ICAPI protocols, the tone IDs for any additional tones that must be redefined after calling the **gc_Attach()** function cannot be in the range from 101 to 189.

3.2. Call Analysis

Call analysis consists of both pre-connect and post-connect information about the status of the call. Pre-connect information (call progress) determines the status of the call connection, that is, busy, no dial tone, no ring back, etc. Post-connect information (media type detection) determines the destination party's media type, that is, answering machine, fax, modem, voice, and the like.

- NOTES:**
1. In Springware terminology, the term call analysis applies to the terms call progress and media detection.
 2. Post-connect information is also known as media type detection.

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 *GlobalCall API Software 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 protocol (.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 *Call Analysis Chapter* in the *Voice Software Reference - Features Guide* for your operating system. For a detailed description of the **gc_LoadDxParm()** function, see the *GlobalCall API Software Reference* for more information.

3. Developing GlobalCall E-1 CAS or T-1 Robbed Bit Applications

Some example uses of call progress tones are as follows:

- By detecting the ringback tone, the GlobalCall 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.

3.2.1. Call Analysis Functionality for ICAPI Protocols

GlobalCall call analysis uses GTD and timers. Some of the country dependent protocol (.cdp) parameters 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 *Global Tone Detection/Generation* chapter in the *Voice Software Reference - Features Guide* for your operating system for information on 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 ICAPI timer expires.

To disable Connect detection, set parameter \$3 to 0. GlobalCall 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 protocol (*.cdp*) file.

Only the call progress tone definitions in the *.cdp* file are used by the GlobalCall API. The R1 and R2 tone definitions are only used if you disable R2MF support in the *icapi.cfg* file by setting the \$17 parameter to 1.

3. Developing GlobalCall E-1 CAS or T-1 Robbed Bit Applications

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

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

See the *Call Analysis* chapter in the *Voice Software Reference – Features Guide* for your operating system for information on 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 country dependent parameter (*.cdp*) file(s) to ensure proper operation of the protocol. Refer to the *GlobalCall Country Dependent Parameters (CDP) 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.

3.2.2. Call Analysis Functionality for PDK Protocols

PDK protocols configure default call analysis operation through the use of two Protocol Service Layer (PSL) parameters in the protocol *.cdp* file:

- **PSL_MakeCall_CallProgress** - 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 a T-1 and an analog protocol. However, the protocol must be 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 GlobalCall. This value is typically used by an E-1 protocol.
- **PSL_MakeCall_MediaDetect** - 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 a T-1 and an analog protocol. The protocol must however be 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 GlobalCall.. This value is typically used by an E-1 protocol. This is the default value if this parameter is left undefined in the *.cdp* file.

3. Developing GlobalCall E-1 CAS or T-1 Robbed Bit Applications

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 GlobalCall. More specifically:

- When the **PSL_MakeCall_CallProgress** 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()**, for example, when using GlobalCall.
- When the **PSL_MakeCall_MediaDetect** parameter in the *.cdp* file is not specified as 1 (Preferred, the default is 2, Pass-through), the application may instead enable media type detection (the default is disabled) in a call to **gc_MakeCall()** or **gc_SetParm()** when using GlobalCall.
- 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 are ignored, for example, the settings specified via **gc_MakeCall()** or **gc_SetParm()** when using GlobalCall.

As mentioned, PDK protocols support call analysis via the **gc_MakeCall()** function, which 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**. 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 enabled/disabled call progress and/or media type detection as required. If this method is used 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. Consequently, the application may need to poll for the information.

PDK protocols also support a more preferred method of media detection using the **gc_SetParm()** function. The parameter used to specify media detection in this case is **GCPR_MEDIADETECT** which can take the values **GCPV_ENABLE** or **GCPV_DISABLE**. 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, that is, the application does not have to wait for a **GCEV_CONNECTED** event.

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**. See the **gc_GetCallInfo()** function description for a list of the values that may be returned when the **info_id** parameter is **CONNECT_TYPE**.

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 utilized 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 T-1 protocols.

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
```

In a SpringWare environment, there are two basic types of tone detection for both single and dual tones; edge-detection and cadence detection.

3. Developing GlobalCall E-1 CAS or T-1 Robbed Bit Applications

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 2* below.

Tone detection using the cadence detection algorithm provides notification when the energy in the specified frequency band(s) exceed 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 desired bounce time
- **On Time Deviation** - minus half the desired bounce 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, Frequency1 +/- Frequency1 Deviation) encompassing both dual tone ranges.

The meaning of each argument is explained in *Table 2*.

Table 2. 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) Note: 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) Note: 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) Note: The minimum recommended value is 50.	Detect, Generate	Cadence

3. Developing GlobalCall E-1 CAS or T-1 Robbed Bit Applications

Parameter Number	Name	Description	Detect/Generate	Edge/Cadence Detection
8	OnTime Deviation	On time deviation (in milliseconds) Note: The minimum recommended value is 50.	Detect	Cadence
9	Off Time	Off duration (in milliseconds) Note: The minimum recommended value is 50.	Detect, Generate	Cadence
10	OffTime Deviation	Off time deviation (in milliseconds) Note: The minimum recommended value is 50.	Detect	Cadence

Parameter Number	Name	Description	Detect/Generate	Edge/Cadence Detection
11	Mode	Detection Notification: <ul style="list-style-type: none"> • 1 for the onset of the tone. This specifies leading edge in edge-detection and onset of cadence detection in cadence detection mode. • 0 for the termination of the tone. This specifies trailing edge in edge detection mode and the termination of the cadence after the specified number of cycles in cadence detection mode. 	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
```

3. Developing GlobalCall E-1 CAS or T-1 Robbed Bit Applications

```
/*  
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
```

3.3. Header Files

In addition to the common GlobalCall header files *gclib.h* and *gcerr.h* that are required irrespective of the technology used, the following header files may also be required when developing applications that use PDKRT and ICAPI protocols:

- *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.

3.4. Resource Association

In E-1 CAS and T-1 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). If a DTI/301SC board (E-1 interface) or a DTI/241SC board (T-1 interface) is used, the tonal part of the call establishment phase is handled by a tone resource. Voice channel, voice resource and tone resource are used interchangeably in this manual when discussing GlobalCall functionality.

Typically, in E-1 CAS or T-1 robbed bit environments, a GlobalCall 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 GlobalCall 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 GlobalCall

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 GlobalCall line device using the **gc_Attach()** function and then routed to the line device's network time slot. The **gc_Attach()** function tells the GlobalCall 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 5.1. Dedicated Voice Resources* and *Section 5.2. Shared Voice Resources* for more information.

3.5. Alarm Handling

As described in the *GlobalCall API Software 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.

When an alarm occurs on a GlobalCall 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).

3.6. Run Time Configuration of the PDKRT Call Control Library

Table 3 shows the parameters of the PDKRT call control library that can be configured using the Real Time Configuration Manager (RTCM). 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.

3. Developing GlobalCall E-1 CAS or T-1 Robbed Bit Applications

NOTE: Since these parameters are statically defined, the `gc_QueryConfigData()` is not applicable.

Table 3. 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 <code>gc_GetCallInfo()</code>)	string	GC_R_O
	CATEGORY_DIGIT	GCTGT_CCLIB_CRN	Category digit type (alternative to <code>gc_GetCallInfo()</code>)	char	GC_R_O
	CONNECT_TYPE	GCTGT_CCLIB_CRN	Connect type (alternative to <code>gc_GetCallInfo()</code>)	char	GC_R_O
GCSET_PARM	GCPR_CALLING_PARTY	GCTGT_CCLIB_CHAN	Calling party (alternative to <code>gc_GetParm()/gc_SetParm()</code>)	string	GC_W_I
	GCPR_LOADTONES	GCTGT_CCLIB_CHAN	Load tones (alternative to <code>gc_GetParm()/gc_SetParm()</code>)	short	GC_W_I
GCSET_ORIG_ADDR	GCPARM_ADDR_DATA	GCTGT_CCLIB_CHAN	Calling number (alternative to <code>gc_SetCallNum()</code>)	string	GC_W_I
<p>Note * GC_W_I - update GC_R_O - retrieve only GC_W_N - update only at null state GC_W_X - not available</p>					

3.7. Run Time Configuration of PDK Protocol Parameters

Configurable PDK protocol parameters are grouped in two sets:

- Protocol State Information (PSI) variable parameters

- Protocol Service Layer (PSL) variable parameters

NOTE: To avoid errors, both PSI and PSL parameters of a GCTGT_PROTOCOL_CHAN channel are only allowed to be changed when the channel object does not have an active call.

Protocol State Information (.psi) variable parameters are interpreted by the PDK run-time component (PDKRT). The names of the Protocol State Information (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_QueryData()** are shown in *Table 4*.

Table 4. CDP Parameters

Parameter Name	Data Type
CDP_ANI_ENABLED	boolean
CDP_ANI_MaxDigits	integer
CDP_CallingPartyCategory	string
CDP_CONNECT_UPON_MEDIA	boolean
CDP_DNIS_DIGITS_BEFORE_ANI	integer
CDP_DNIS_DIGITS_BEFORE_CAT	integer
CDP_DNIS_ENABLED	boolean
CDP_DNIS_MaxDigits	integer
CDP_IMMEDIATE_ACCEPTSTATE	boolean
CDP_NUM_OF_ANI_DIGITS	integer
CDP_NUM_OF_DNIS_DIGITS	integer

The Protocol Service Layer (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. 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.

3. Developing GlobalCall E-1 CAS or T-1 Robbed Bit Applications

In the latter case, the parameters will most likely have a platform tag. All of these parameter names must begin with "PSL_". The names of the Protocol Service Layer (PSL) variable parameters begin with PSL_ and SYS_. The PSL parameters that can be accessed via `gc_GetConfigData()`, `gc_SetConfigData()`, and `gc_QueryData()` are shown in *Table 5*.

Table 5. PSL and SYS Parameters

PSL Variable Name	Data Type
PSL_AcceptCallDefaultNumOfRings	Integer
PSL_AnswerCallDefaultNumOfRings	Integer
PSL_MakeCall_CallProgress	Integer
PSL_MakeCall_MediaDetect	Integer
PSL_TONE_CP_BUSY1	Tone
PSL_TONE_CP_BUSY2	Tone
PSL_TONE_CP_DIAL_INTL	Tone
PSL_TONE_CP_DIAL_LCL	Tone
PSL_TONE_CP_DIAL_XTRA	Tone
PSL_TONE_CP_FAX1	Tone
PSL_TONE_CP_RNGBK1	Tone
PSL_TONE_CP_RNGBK2	Tone
PSL_DefaultNumOfRingsforAcceptCallandAnswerCall	Integer
PSL_DefaultMakeCallTimeout	Integer
PSL_ANALOG_NUM_RINGS_BEFORE_RINGON	Integer
PSL_DXCAS_HOOKFLASH_DURATION	Integer
SYS_FEATURES	String
SYS_PSINAME	String

Table 6 shows the Set ID and Parm ID for these parameter types.

Table 6. Configurable PDK Protocol Parameters

Set ID	Parm ID	Target Object Type	Explanation	Update Flag
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
<p>Note: * indicates that CAS pattern signals and tones cannot be accessed. ** GC_W_I - update GC_R_O - retrieve only GC_W_N - update only at null state GC_W_X - not available</p>				

The PDK GCTGT_PROTOCOL_SYSTEM target object is not available until the first **gc_OpenEx()** function is called to run this protocol.

The GlobalCall 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 GlobalCall application must call **gc_QueryConfigData()** to find the parameter information (set ID, parm ID, and value data type, etc.) first. For more information on these functions, refer to the *GlobalCall Application Developer's 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, GC protocol ID)

3. Developing GlobalCall E-1 CAS or T-1 Robbed Bit Applications

- (GCTGT_PROTOCOL_CHAN, GlobalCall 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_ParmIDstruct; /* 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_ParmIDstruct.set_ID,
                      t_ParmIDstruct.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);
```

3.8. Determining the Protocol Version

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

GlobalCall™ E-1/T-1 Technology User's Guide for Linux and Windows

```
#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_dtiBlTl:V_dxxxBlC1", 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.evttyp == GCEV_UNBLOCKED)
        {
            if (gc_GetParm(ldev, GCPR_PROTVER, &parm) ==
                GC_SUCCESS)
            {
                printf("The protocol version: %s\n", parm.paddress);
            }
            else
            {
                // Error processing
                int gc_error;
                int cclibid;
                long cc_error;
                char* gc_msg;
                char* cc_msg;

                gc_ErrorValue(&gc_error, &cclibid, &cc_error);
                gc_ResultMsg(LIBID_GC, (long)gc_error, &gc_msg);
                gc_ResultMsg(cclibid, cc_error, &cc_msg);
                printf("gc_GetParm(GCPR_PROTVER) failed! GC(0x%x) -
                    %s; CC(0x%x) - %s\n",
                    gc_error, gc_msg, cc_error, cc_msg);
                return (gc_error);
            }
        }
    }
}

gc_Close(ldev);
```

3. Developing GlobalCall E-1 CAS or T-1 Robbed Bit Applications

```
gc_Stop();  
return(0);  
}
```


4. Applying GlobalCall Functions to E-1 CAS or T-1 Robbed Bit Applications

Certain GlobalCall functions have additional functionality or perform differently when used in an E-1 CAS or a T-1 robbed bit environment. The general function descriptions in the *GlobalCall API Software Reference* do not contain detailed information on a particular technology. Detailed information in terms of the additional functionality or the difference in performance of those functions in an E-1 CAS or a T-1 robbed bit environment is contained in this chapter. Note that this information must be used in conjunction with the information presented in the *GlobalCall API Software Reference*.

The following sections describe how the functions are used specifically in E-1 CAS or T-1 robbed bit applications; the functions are presented alphabetically. See the *GlobalCall API Software Reference* for detailed function descriptions.

4.1. gc_AcceptCall()

The **gc_AcceptCall()** function uses the **rings** parameter to specify the number of rings to wait before terminating the function, that is, before the GlobalCall API sends the GCEV_ACCEPT event to the application.

For ICAPI protocols, if the **rings** parameter is set to 0, the value specified in \$9 of the country dependent parameters (.cdp) file is used.

For PDK protocols, if the **rings** parameter is set to 0, the value of the **PSL_DefaultNumOfRingsforAcceptCallandAnswerCall** parameter in the country dependent parameters (.cdp) file is used.

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.

4.2. **gc_AnswerCall()**

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 ICAPI protocols, if the **rings** parameter is set to 0, the value specified in parameter \$9 of the country dependent parameter (*.cdp*) file is used.

For PDK protocols, if the **rings** parameter is set to 0, the value of the **PSL_DefaultNumOfRingsforAcceptCallandAnswerCall** parameter in the country dependent parameters (*.cdp*) file is used.

4.3. **gc_CallAck()**

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 *GlobalCall API Software 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).

NOTE: See the *GlobalCall Country Dependent Parameters (CDP) Reference* for limitations in the use of this function.

4. Applying GlobalCall Functions to E-1 CAS or T-1 Robbed Bit Applications

4.4. gc_BlindTransfer()

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 *GlobalCall Country Dependent Parameters (CDP) Reference for PDK Protocols* for more information.

4.5. gc_Close()

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.

4.6. gc_CompleteTransfer()

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 *GlobalCall Country Dependent Parameters (CDP) Reference for PDK Protocols* for more information.

4.7. gc_Detach()

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 GlobalCall call state is Null or Idle. Furthermore, the **gc_Detach()** function can only be called in the Null, Idle, or Connected states.

4.8. gc_DropCall()

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

CAUTION

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 *GlobalCall Country Dependent Parameters (CDP) Reference* 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, the **gc_DropCall()** function occasionally results in the generation of the GCEV_DROPCALL 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.

4. Applying GlobalCall Functions to E-1 CAS or T-1 Robbed Bit Applications

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 GlobalCall 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.

4.9. gc_Extension()

The **gc_Extension()** function provides the capability to dynamically configure the behavior of a PDK protocol in a manner that is specific to that protocol. The **gc_Extension()** function is only valid for applications using PDK protocols that support this feature. Check the associated *.cdp* file or the *GlobalCall Country Dependent Parameters (CDP) Reference for PDK Protocols* for more information regarding the support and configuration of this feature including its usage.

For PDK protocols which support this feature, up to a maximum of three integer values and two strings may be passed to the protocol by configuring the **parm** parameter as a pointer to a structure of type PDK_EXT_SIG_BLK.

```
struct pdk_ext_sig_blk {
    int int1Val;
    int int2Val;
    int int3Val;
    char *str1Val;
    char *str2Val;
} PDK_EXT_SIG_BLK;
```

The actual meaning and usage of these integers and string values are protocol specific. The PDKRT library only supports this function in asynchronous mode (EV_ASYNC), and therefore the **retval** parameter is ignored.

If the protocol requires unsolicited notification of the application as a result of the completion of a feature triggered by the **gc_Extension()** function, or for any other asynchronous notification for any documented reason, it may send an Extension event:

4.10. Extension Event

The extension event, GCEV_EXTENSION, is provided to support unsolicited events resulting from PDK protocol-specific features. Refer to the associated *GlobalCall Country Dependent Parameters (CDP) Reference for PDK Protocols* for the proper handling of any unsolicited extension events as well as the format of the event, that is, identifier mappings and contents of the data buffer.

For this event, the pointer to the extended event data block, *exteventdatap*, within the METAEVENT structure, points to a structure, EXTENSIONEVTBLK, defined as follows:

```
typedef struct{
    unsigned char ext_id;          /* unused for PDK protocols */
    GC_PARM gcparm;              /* GC_PARM union to provide technology-specific info */
    short int parm;
} EXTENSIONEVTBLK;
```

The **gcparm** field of the METAEVENT structure itself points to a structure of type PDK_EXT_SIG_BLK (above) so that three integers and two string values may be returned from the protocol to the application to indicate protocol-specific status information.

The extension block structure, EXTENSIONEVTBLK, referenced in a GCEV_EXTENSION event has a persistence only until the next call of the **gc_GetMetaEvent()** function. In other words, any information contained or referenced in a GCEV_EXTENSION event must be either processed or cached within the application, or risk being lost upon the next call of the **gc_GetMetaEvent()** function.

4.11. gc_GetCallInfo()

The **gc_GetCallInfo()** function can be used to retrieve the CATEGORY_DIGIT parameter for E-1 CAS calls.

In addition, for both E-1 CAS and T-1 robbed bit protocols that support enhanced call analysis (call progress), the **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

4. Applying GlobalCall Functions to E-1 CAS or T-1 Robbed Bit Applications

- 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 Applicable

For E-1 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 *GlobalCall Country Dependent Parameters (CDP) Reference* for applicability.

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 Applicable).

PDK protocols provide support for enhanced call analysis.

4.12. gc_GetCallProgressParm()

The **gc_GetCallProgressParm()** function is only valid for applications that use PDK protocols. (See the *GlobalCall API Software Reference* for a complete listing of GlobalCall functions and for detailed function descriptions.)

4.13. gc_HoldCall()

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 *GlobalCall Country Dependent Parameters (CDP) Reference for PDK Protocols* for more information.

4.14. gc_MakeCall()

When using E-1 CAS or T-1 robbed bit line devices, the **timeout** argument in the **gc_MakeCall()** function is supported when the **mode** parameter is set to either EV_SYNC or EV_ASYNC.

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

- If the **timeout** argument is set to 0, then the time-out and ringback parameters in the .cdp file are used to set the timeout conditions. Refer to *Section 6.3. Protocol Components* for more information about the .cdp file.
- 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 country dependent parameter (.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 *Function Reference* chapter of the *GlobalCall API Software Reference*.
- If all **timeout** values are set to 0, no timeout condition will apply.

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

- The **timeout** argument in the **gc_MakeCall()** function.
- The **PSL_DefaultMakeCallTimeout** parameter specified in the .cdp file if the timeout argument 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 argument in the **gc_MakeCall()** function is 0 and call analysis is specified.

NOTE: PDK protocols do not use the outbound number of ringback tones to define the timeout.

If your T-1 robbed bit circuit is provisioned for Feature Group A, your application should call the **gc_MakeCall()** function with a null dial string.

If a protocol error occurs during dialing and the default call progress enabled governs, (see *Section 4.14.1. IC_MAKECALL_BLK*) then an error code or an event is returned as described in the **gc_MakeCall()** function description in the *GlobalCall API Software Reference*. If call progress is disabled and a protocol

4. Applying GlobalCall Functions to E-1 CAS or T-1 Robbed Bit Applications

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 transit to the Alerting state
- if voice is detected, to transit 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. See the *Programming Tips for Drop and Insert Applications* section in *Chapter 3* of the *GlobalCall API Software Reference* for details.

To change the enabled call progress default when making a call, see *Section 4.14.1. IC_MAKECALL_BLK* for ICAPI protocols and *Section 4.14.2. PDK_MAKECALL_BLK* for PDK protocols.

4.14.1. IC_MAKECALL_BLK

For ICAPI protocols, 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, see *Table 7. IC_MAKECALL_BLK Field Descriptions*.

The IC_MAKECALL_BLK structure contains information used by the **gc_MakeCall()** function when setting up a call. The structure is defined as follows:

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

Table 7. IC_MAKECALL_BLK Field Descriptions

Field	Description
flags	Controls call analysis and media type detection on a per call basis. The flags included are: <ul style="list-style-type: none"> • 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.

4.14.2. PDK_MAKECALL_BLK

For PDK protocols, 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, see *Table 8. PDK_MAKECALL_BLK Field Descriptions*.

The PDK_MAKECALL_BLK structure contains information used by the **gc_MakeCall()** function when setting up a call. The structure is defined as follows:

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

Table 8. PDK_MAKECALL_BLK Field Descriptions

Field	Description
flags	Controls call analysis and media type detection on a per call basis. The flags included are: <ul style="list-style-type: none"> • NO_CALL_PROGRESS - Set to 0 to enable call analysis (default). Set to 1 to disable call analysis. • MEDIA_TYPE_DETECT - Set to 0 to enable media type

4. Applying GlobalCall Functions to E-1 CAS or T-1 Robbed Bit Applications

Field	Description
	detection. Set to 1 to disable media type detection.
*v_rfu_ptr	Reserved for future use.
ul_rfu[4]	Reserved for future use.

4.15. gc_OpenEx()

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 E-1 or T-1 feature of this function specifies the voice device as part of the **devicename** parameter.

4.15.1. 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 conventions described below allow the GlobalCall API to map subsequent calls made on specific line devices or CRNs to interface-specific libraries. The fields within the **devicename** parameter must each begin with a colon.

The **<network_device_name>** and the **<protocol_name>** are required. The **<voice_channel_name>** is optional depending on your application (see *Section 5.1. Dedicated Voice Resources* or *Section 5.2. Shared Voice Resources*).

- **<network_device_name>** - 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_named> - Specifies the protocol to use. For ICAPI and PDK protocols, use the root file name of the country dependent parameter (.cdp) file.
- <voice_channel_name> - 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>

4.15.2. Examples of the devicename Parameter

The following examples illustrate the **devicename** parameter when using E-1 ICAPI protocols:

- For a D/300SC-E1 board specified as dtiB3, the **devicename** for time slot 1 on this board using the inbound Brazil protocol is:

```
:N_dtiB3T1:P_br_r2_i
```

- To open both voice channel 2 on virtual voice board 4 and the above network device, the **devicename** for the inbound Brazil protocol is:

```
:N_dtiB3T1:P_br_r2_i:V_dxxxB4C2
```

- A line device may represent a board, thus to open the board dtiB1, the **devicename** is:

```
:N_dtiB1:P_br_r2_i
```

4.15.3. Other gcOpen() and gc_OpenEx() Considerations

For E-1 CAS or T-1 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 E-1 CAS or T-1 robbed bit operations. When a voice resource is specified, GlobalCall automatically opens the voice device and internally attaches the voice device to the line device.

4. Applying GlobalCall Functions to E-1 CAS or T-1 Robbed Bit Applications

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 5.1. Dedicated Voice Resources* and *Section 5.2. Shared Voice Resources* 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.

4.16. gc_RetrieveCall()

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 *GlobalCall Country Dependent Parameters (CDP) Reference for PDK Protocols* for more information.

4.17. gc_SetBilling()

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 *GlobalCall Country Dependent Parameters (CDP) Reference* for protocol specific limitations.

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

4.18. gc_SetCallProgressParm()

The **gc_SetCallProgressParm()** function is only valid for applications that use PDK protocols. See the *GlobalCall API Software Reference* for more information.

4.19. gc_SetChanState()

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 E-1 CAS or T-1 robbed bit environment.

4.20. gc_SetEvtMsk()

For ICAPI protocols, the following mask parameter values are supported:

- GCMSK_ALERTING
- GCMSK_BLOCKED
- GCMSK_UNBLOCKED

For PDK protocols, all of the mask parameter values are supported. See the **gc_SetEvtMsk()** function reference page in the *GlobalCall API Software Reference* for more information.

4. Applying GlobalCall Functions to E-1 CAS or T-1 Robbed Bit Applications

4.21. gc_SetParm()

The **gc_SetParm()** function sets the default parameters and all channel information associated with the specific line device. The **GCPR_LOADTONES** parameter enables (set **GCPV_ENABLE** flag) or disables (set **GCPV_DISABLE** flag) downloading of predefined call progress tones to the firmware. These tones are predefined in the E-1 CAS or T-1 robbed bit specific configuration files (see *Section 3.1. GTD Tone Considerations* for details) and are used for call analysis. They are downloaded during execution of the **gc_Attach()** function.

Applications that use application-defined tones or that seek to inhibit tones from being downloaded with each **gc_Attach()** function should use **GCPV_DISABLE** to disable the **GCPR_LOADTONES** parameter.

4.22. gc_Start() and gc_Stop()

For 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 8.1. Debugging Applications that use ICAPI Protocols* for more on log files.

For PDK Protocols, the **gc_Start()** function is used to access the error and debug logging capabilities of the PDKRT call control library. See *Section 8.2. Debugging Applications that use PDK Protocols* for more information.

4.23. gc_StartTrace()

For 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. See *Section 4.22. gc_Start() and gc_Stop()* for more information.

For PDK protocols, the **filename** argument is ignored. The name of the log file is specified in the PDK_START_STRUCT data structure. See *Section* for more information.

4.24. gc_SetupTransfer()

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 *GlobalCall Country Dependent Parameters (CDP) Reference for PDK Protocols* for more information.

4.25. gc_SwapHold()

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 *GlobalCall Country Dependent Parameters (CDP) Reference for PDK Protocols* for more information.

5. Resource Allocation and Routing

E-1 CAS and T-1 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 E-1 CAS and T-1 robbed bit environment are discussed in this chapter.

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. GlobalCall simplifies applications written to handle E-1 CAS and T-1 robbed bit protocols using dedicated voice resource configurations. To use GlobalCall functionality to setup dedicated resources, the application must pass both the network time slot and the voice channel to the **gc_OpenEx()** function. The GlobalCall API uses this information to automatically:

- open the network board device, if not previously opened (the board device is used internally by GlobalCall)
- open both the voice channel and the network time slot
- route the voice channel and network time slot together (full duplex) (CTbus configurations only)
- associate the voice channel and the network time slot by issuing an internal **gc_Attach()** function

For CTbus 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.

5.1.1. Dedicated Voice Resources Example

```

.
#define MAXCHAN 30
struct linebag{
    LINEDEV  ldev;
    CRN      crn;
    INT      state;
}port[MAXCHAN+1]
.
/* Open a GlobalCall 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 GlobalCall line device using time slot 1 of dtiB1 opens voice channel dxxxB1C1 configures 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.

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 (E-1 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 GlobalCall 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 CTbus **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 GlobalCall 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.

5.2.1. Shared Voice Resources - LINUX and Windows 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("dxoB1C1", 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_scuroute(networkh, SC_DTI, voiceh, SC_VOX, SC_FULLDUP) == -1)
          {
              /* process error */
          }

```

Legend:

- 1 The `gc_OpenEx()` function:
 - opens a GlobalCall line device using time slot 1 of dtiB1 using outbound Brazilian R2 protocol.

GlobalCall™ E-1/T-1 Technology User's Guide for Linux and Windows

- 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 GlobalCall line device.
- 8 The **nr_scunroute()** function unroutes the voice and network resources.

6. Protocols

The protocols supported, the protocol naming conventions, the protocol components and their corresponding protocol files are described in the following sections.

Descriptions of the parameters in the country dependent parameter (*.cdp*) files that are especially relevant to a protocol are described in the *GlobalCall Country Dependent Parameters (CDP) Reference*.

6.1. Protocols Supported

Protocols are distributed separately on individual disks or as part of a software package release. This modular design simplifies adapting applications for use in numerous countries. 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 *GlobalCall Country Dependent Parameters (CDP) Reference* for additional information.

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, Dialogic provides *.cdp* country dependent parameter files that can be modified to satisfy local requirements. Both inbound and outbound protocols used on a trunk must use the same *.prm* parameter file.

The GlobalCall protocols available are listed in the *GlobalCall Country Dependent Parameters (CDP) Reference*. For the most up-to-date list of available protocols, contact your nearest Dialogic Sales Office. For Sales Offices and other contact information, visit our website at <http://www.dialogic.com>.

6.2. Protocol Naming Convention

The GlobalCall protocol files use the naming convention described in *Table 9*.

Table 9. Protocol Naming Convention

Code	Description
ccl	indicates the call control library for which the protocol is written, for example, pdk represents the PDKRT call control library. For the ICAPI and ANAPI call control libraries, ccl is blank.
cc	2 character ISO country code; for example, es = Spain, fr = France and mx = Mexico
tt	2 character technology type. Valid types are: <ul style="list-style-type: none"> • em: T-1 protocol using E&M signaling with support for DTMF digits only • mf: T-1 protocol using E&M signaling with support for MF digits • r2: protocol using R2 MFC signaling • r1: protocol using R1 MFC signaling • e1: pulse, MF SOCOTEL or other E-1 protocol • ls: loop start protocol
ffff (optional)	Defines a special software or hardware feature supported by the protocol; 1 to 4 characters. NOTE: Requires ICAPI call control library level 2, else a compatibility error, EGC_COMPATIBILITY, will be generated when the application attempts to load the protocol.
d	1 or 2 character direction indicator. Valid directions are: <ul style="list-style-type: none"> • i - inbound • o - outbound • io - inbound/outbound
nn	2 character code indicating the number of time slots per trunk: <ul style="list-style-type: none"> • 24 for T-1

Code	Description
	<ul style="list-style-type: none"> • 30 for E-1

The GlobalCall protocol component names are constructed as described in the following table.

Table 10. Protocol Component Name Structure

Code	Description
cc_tt_d.so or cc_tt_ffff_d.so	ICAPI protocol module for LINUX
cc_tt_d.dll or cc_tt_ffff_d.dll	ICAPI protocol module for Windows
pdk_cc_tt_d.psi	PDK protocol state information file
cc_pr_d.cdp or cc_tt_ffff_d.cdp	country dependent parameter file for LINUX or Windows
pdk_cc_pr_d.sd p or pdk_cc_tt_ffff_ d.sdp	PDK protocol site dependent parameter file
cc_nn0.prm	voice and network parameters

The protocol name used in the **devicename** parameter of the **gc_OpenEx()** function is the root name of the *.cdp* file. Most ICAPI protocol releases use separate protocol modules to handle the inbound and the outbound portions of a protocol. For example, *Table 11* describes the files included for the Argentina R2 ICAPI protocol.

Table 11. 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>
Voice and network firmware parameters	<i>ar_300.prm</i>	<i>ar_300.prm</i>

For PDK protocols, we recommend that you use bi-directional protocols. For example, *Table 12* describes the files included with the Argentina R2 PDK protocol.

Table 12. Sample PDK Protocol File Set

Description	Protocol Files
	LINUX and Windows
Bi-directional protocol module	<i>pdk_ar_r2_io.psi</i>
Bi-directional country dependent parameters	<i>pdk_ar_r2_io.cdp</i>
Voice and network firmware parameters	<i>ar_300.prm</i>

NOTE: For PDK protocols, you can create a site dependent parameter (*.sdp*) file, which defines site-specific information.

6.3. Protocol Components

The following files are included on a protocol CD-ROM:

- **protocol modules:** these files contain protocol specific information and are dynamically linked to the application as needed. For PDK protocols in either LINUX or Windows environments, the protocol module is a protocol state information (*.psi*) file, a binary file that is interpreted by the PDK run-time component (PDKRT).
- **country dependent parameter (.cdp) file:** this file contains protocol related parameters. In addition to the firmware parameter (*.prm*) files, GlobalCall

uses a country dependent parameter (*.cdp*) file that defines country specific and protocol specific parameters for use by GlobalCall.

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.

- **site dependent parameter (.sdp) file:** this file has the same format as the *.cdp* file. The *.sdp* file is not supplied with the protocol but can be created by the user to define local information, for example, local telephone numbers as well as define switch-dependent variants and implement variants of the same basic protocol.
- **standard voice and network firmware parameter file(s) for the country:** these parameters have a file extension *.prm* and are located:
 - in the */usr/dialogic/data* directory for LINUX
 - in the *\Program Files\Dialogic\data* directory in Windows (**NOTE:** the *\Program Files\Dialogic* directory is the default directory. When installing the Dialogic System Software and SDK, a different directory may be specified.)

The filename(s) have the same naming convention and format as the *.prm* files included in the Country Specific Parameter package that accompanies Dialogic System Software.

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

6.3.1. Country Dependent Parameter (.cdp) Files

Descriptions of the country dependent parameters most likely to be modified for a protocol are provided in the *GlobalCall Country Dependent Parameters (CDP) Reference*. In the following discussions, *ccl* indicates the call control library for which the protocol is written. For example, *pdk* represents the PDKRT call control library. For the ICAPI and ANAPI call control libraries, *ccl* is blank. *cc* represents the 2-character country code. For example, *in_r2_o.cdp* would be for the India R2 protocol file.

Country dependent parameter (.*cdp*) files may be customized by modifying the files for the inbound protocol module (for example, *ccl_cc_r2_i.cdp*) or for the outbound protocol module (for example, *ccl_cc_r2_o.cdp*).

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

For Windows: *x:\Program Files\Dialogic\data*

where **x** is a drive name.

For LINUX: */usr/dialogic/cfg*

6.3.2. PDK Site Dependent Parameter (.sdp) Files

PDK protocols can use site dependent parameter (.*sdp*) files. An .*sdp* file, which has the same format as a country dependent parameters (.*cdp*) file, can be created to define site-specific parameters, such as local phone numbers and switch-dependent variables. The parameter definitions in the .*sdp* file override any of the corresponding parameter definitions in the .*cdp* file, but may also include additional parameters that are supported, but not in the .*cdp* file.

By defining local parameters, such as local phone numbers and switch-dependent variables in an .*sdp* file, loss of these parameter definitions can be avoided if the protocol and/or .*cdp* file are updated.

If a .*sdp* file is being used, and a change is made to one or more parameters in the .*sdp* file, the application must be restarted before any parameter changes take effect. See *Appendix B* for a sample .*sdp* file.

NOTE: ICAPI protocols do not support .*sdp* files.

The .*sdp* file should be located **only** under the directory:

For Windows: *x:\Program Files\Dialogic\data* where **x** is a drive name.

For LINUX: */usr/dialogic/cfg*

6.3.3. Parameter (.prm) Files (LINUX only)

For some protocols, certain parameters must be specified in the firmware parameter (.prm) file to ensure proper operation of the protocol. Refer to the *GlobalCall Country Dependent Parameters (CDP) Reference* for country dependent parameters for any required settings.

When the protocol package is installed on your system, the *cc_nn0.prm* file is installed. The *cc_nn0.prm* file contains both the voice and network parameters required by the GlobalCall API. You must set the Country parameter for LINUX:

- in the */usr/dialogic/cfg/dialogic.cfg* file for each board to specify the *cc_nn0.prm* file as shown below

```
ParameterFile = cc_nn0.prm
```

where,

- **cc** is the country code
- **nn** is the number of time slots per trunk.

For some protocols, different firmware parameters may be needed to support different variants of the protocol. See the *GlobalCall Country Dependent Parameters (CDP) Reference* for specific instructions regarding parameter selection. To use parameter values in the *ParamFile* file that differ from the default values, you must set the **ParamFeature** parameter in the */usr/dialogic/cfg/dialogic.cfg* file for each board so as to specify the protocol variant to use.

6.3.4. Parameter (.prm) Files (Windows only)

For some protocols, certain parameters must be specified in the firmware parameter (.prm) file to ensure proper operation of the protocol. Refer to the *GlobalCall Country Dependent Parameters (CDP) Reference* for any required settings.

When the protocol package is installed on your system, the *cc_nn0.prm* file is installed. This parameter (.prm) file contains both the voice and network parameters required by the GlobalCall software.

Use the Dialogic Configuration Manager utility as described in the Dialogic System Installation documentation to configure each board in your system:

- select the board to be configured
- select the Country tab
- select the country name from the pull down menu
- use the Edit window to change parameter values for the board and country selected

For some protocols, different firmware parameters may be needed to support different variants of the protocol. See the *GlobalCall Country Dependent Parameters (CDP) Reference* for specific instructions regarding parameter selection. To use parameter values in the *cc_nn0.prm* file that differ from the default values, use the Dialogic Configuration Manager to set the firmware parameters to the desired value.

6.4. Using ICAPI and PDK Protocols

The following are the possible scenarios:

- **Using only ICAPI protocols.** - You must backup and remove PDK protocols and corresponding *.cdp* files, then install the ICAPI protocols you need and set the `PREFER_ICAPI` system-wide environment variable.
- **Using only PDK protocols.** - You must backup ICAPI protocol *.cdp* files, if necessary, then install the PDK protocols you need and ensure that that `PREFER_ICAPI` system-wide environment variable is not set.
- **Using an ICAPI protocol and a different PDK protocol at the same time.** - This is a valid option. You can install and use both protocols.

- **Using an ICAPI protocol and a PDK protocol of the same name at the same time.** - This is not a valid option. You must use either the ICAPI protocol or the PDK protocol. Either option requires renaming the other protocol and the corresponding *.cdp* file, and then reloading a protocol or restoring it from backup.

The rules governing the selection of a protocol type are as follows:

1. If a *.psi* file exists, then the PDK protocol is used, otherwise, the ICAPI protocol is used.
2. If the environment variable PREFER_ICAPI is set, rule 1 above is overridden.

NOTE: Both ICAPI and PDK protocols use a different *.cdp* file format.

7. ICAPI Configuration Parameters

The parameters shown below are available in the *icapi.cfg* file. Unless otherwise instructed, these parameters should retain their original settings.

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

- for LINUX: */usr/dialogic/cfg*
- for Windows: *\Program Files\Dialogic\cfg*

For Windows applications, the log file (*icapi.log.<pid>*, where pid = the process identification number) is generated by setting the parameters \$11 and \$12 in the *icapi.cfg* file as indicated in the following table.

Table 13. *icapi.cfg* File Parameters

Parameter	Description
\$11	<p>Logging utility (default = 0):</p> <ul style="list-style-type: none"> • Set to 0 to ignore parameters \$12, \$13 and \$15. • Set to 1 to enable logging, either to the screen (set \$13 parameter to 1) or to the <i>icapi.log.<pid></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. <p>NOTE: Enabling logging is not recommended during normal operation due to the increased host processor loading.</p> <ul style="list-style-type: none"> • (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.
\$12	<p>Number of the channel to be monitored (default = 0):</p> <ul style="list-style-type: none"> • A value of 0 means monitor all opened devices. • A value of -1 means do not monitor any device. • Entering a channel number designates the channel to be monitored.
\$13	<p>Echo on screen (default = 0):</p> <ul style="list-style-type: none"> • Set to 0 to ignore parameter. • Set to 1 to send the debug information to the screen.
\$14	<p>Disable DTI Wait Call function (default = 0):</p> <ul style="list-style-type: none"> • The 0 default selection causes the DTI Wait Call firmware function to wait for an incoming call at the board firmware level. • The 1 selection causes the DTI Wait Call firmware function to wait for an incoming call at the ICAPI call control library level. <p>The value selected is protocol-dependent; do not change the default value unless instructed to do so in the documentation for</p>

7. ICAPI Configuration Parameters

Parameter	Description
	your protocol.
\$15	(LINUX only) Size of debug memory (default = 1; that is, 1 = 1 event or action in memory): 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. <ul style="list-style-type: none">• Set to 1 to store one action or event in the buffer.• Set to 0 to ignore feature (default).
\$18	Enables cadenced tones, such as ringback and busy, to be played using the firmware rather than using host-based function calls such as dx_playtone() and sleep() . <ul style="list-style-type: none">• Set to 0 to disable firmware cadence tones (default).• Set to 1 to enable firmware cadence tones.

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

\$16 and \$17 are for backwards compatibility only and should not be changed.

GlobalCall™ E-1/T-1 Technology User's Guide for Linux and Windows

8. Debugging Applications

GlobalCall 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. The GlobalCall debugging utilities are described in this chapter.

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

8.1. Debugging Applications that use ICAP Protocols

For LINUX applications, the log file (*icapi.log.<pid>*, where *pid* = the process identification number) 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 ICAP log file, use the `rs_log_printf()` function. This function works like the `fprintf()` function except that a file descriptor is not used.

8.2. Debugging Applications that use PDK Protocols

In a SpringWare environment, the GlobalCall 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.

8.2.1. Enabling and Disabling the Logging

CAUTION:

We recommend that logging be done on an as-needed basis. Logging uses significant resources and can reduce the performance of the GlobalCall 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.

The LogView tool is required to view the log file.

In an environment that uses SpringWare boards, 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 4.22. gc_Start() and gc_Stop()* and *Section 4.23. gc_StartTrace()* for more information.

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

- Populating and using a CCLIB_START_STRUCT
- Defining the GC_PDK_START_LOG environment variable

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.

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;
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 14*.

Table 14. cclib_data Fields and Values

Field Name	Field Values	Default Value
filename	Log file name	gc_pdk.log
loglevel	See <i>Table 15</i> .	ENABLE_FATAL or 5
cachedump	See <i>Table 17</i> .	WHEN_FULL or 1
cachesize	Any positive integer	1 (number of records in cache)
channel	See <i>Table 18</i> .	B*C*

Field Name	Field Values	Default Value
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 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 a start_parameters string. In practice, you only need to specify the values of fields that are different than the default values.

```
cclib_data = "filename: pdktest.log;
binaryfile: 1;
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 tables following show the allowable values for the **loglevel**, **service**, **cachedump** and **channel** fields respectively. The values of **loglevel**, **service** and **cachedump** can be numbers (if hex format is used, the prefix 0x should be used) or symbols. 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 15 shows the valid values the log_level parameter.

Table 15. log_level Values

log_level	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> , dx_open() 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. For example, in <i>globalcallload.cpp</i> , if the <i>.sdp</i> file does not exist, then a log record with ALERT rather than WARNING will be issued.
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 gc_StartTrace() and gc_StopTrace() determines if logging for a specific channel is on or off. This kind of information is a level higher than DEBUG.

log_level	Valid Value	Description
ENABLE_DEBUG	1	All levels are logged. The lowest level with 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.
Note: Values are in decimal but can also be specified in hex using a 0x prefix.		

Table 16 shows the valid values the **log_service** parameter.

Table 16. log_service Values

log_level	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 R2MF 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.
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 (65546)	Only GCLIB service enabled.

log_level	Valid Value	Description
Note: Values prefixed with 0x are hexadecimal values. Decimal values are shown in parenthesis.		

Table 17 shows the valid values for the log_cachedump parameter.

Table 17. log_cachedump Values

log_level	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 cachesize parameter. For example, if cachesize 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.
Note: Values prefixed with 0x are hexadecimal values.		

Table 18 shows some examples of the log_channel parameter.

Table 18. Sample log_channel 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.

8. Debugging Applications

Example Value	Boards and Channels Enabled for Logging
B1C1	Channel 1 on board 1.
B1C1-5	Channel 1 to 5 on board 1.
B1C1,20	Channel 1 and 20 on board 1.
B1-4C*	All channels of board 1 to 4.
B1C2, B2C2,20-22	Channel 2 of board 1, channel 2, 20, 21, and 22 on board 2.

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 the GC_PDK_START_LOG environment variable:

- The following is an example of a GC_PDK_START_LOG environment variable definition:

```
set GC_PDK_START_LOG="filename : pdktest.log;  
loglevel: ENABLE_DEBUG;  
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 the *Populating and Using a CCLIB_START_STRUCT* section immediately above and the definition for each field is also the same as described in that section.

NOTE: The example above shows all the possible fields values in the environment variable. In practice, you only need to specify the values of fields that are different than the default values.

GlobalCall™ E-1/T-1 Technology User's Guide for Linux and Windows

Appendix A - Related Publications

This appendix lists publications you should refer to for additional information on E-1 or T-1 telephony. For additional information about Dialogic products, visit our web site or refer to the Dialogic publications listed in the *Related Publications Appendix* in the *GlobalCall API Software Reference*.

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

T-1 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

Appendix B - Sample .sdp File

This appendix provides an example of a *.sdp* file that can be used with PDK protocols. The parameters defined in the *.sdp* file override parameters that have the same name in the corresponding *.cdp* file.

```
/**
*****
FILE      : AR_R2_IO.SDP
Author    : John Doe, Customer Site
*****
HISTORY
*****
25 April 1999 Rel 1.0  Creation

21 May 1999 Rel 2.0   Set the localNumber to "5551212"

15 July 1999 Rel 3.0  Enabled the CDP_ANI_ENABLED parameter
                      to receive ANI digits.

3 Sept 1999 Rel 3.1   Set the CDP_MAKECALLTIMEOUT parameter
                      to 30 seconds.
*****

This is a sdp file that is to be used with Argentina PDK Protocol.
The parameters specified in the CDP file will be over-written by
the parameters included in this file.

Ensure that this file is in x:\program files\diallogic\cfg directory.
where x is the drive (C or D) where Dialogic PDKRT is installed.

*/

/*
*****
CDP_LocalNumber
*****
This parameter sets the local number
*/

ALL Charstring_t CDP_LocalNumber = "5551212"

/*
*****
CDP_DNIS_ENABLED
*****
Inbound

This parameter should be set to 1 if DNIS digits are to be received
else set this parameter to 0.

Note :
o If this parameter is set to value other than 0 or 1,
  the behaviour of protocol will be undetermined
```

GlobalCall™ E-1/T-1 Technology User's Guide for Linux and Windows

- o Even if this parameter is set to 0, The first forward tone being received will be First DNIS digit only

*/

All BOOLEAN_t CDP_DNIS_ENABLED = 1

/*

CDP_ANI_ENABLED

Inbound

This parameter should be set to 1 if ANI digits are to be received else set this parameter to 0.

Note :

If this parameter is set to value other than 0 or 1, the behaviour of protocol will be undetermined

*/

All BOOLEAN_t CDP_ANI_ENABLED = 1

/*

CDP_DNIS_DIGITS_BEFORE_ANI

Inbound

This parameter determines how many DNIS Digits are to be received Before ANI, Set this parameter to 0, if ANI digits have to be received after Complete DNIS

If set to non-zero value, following sequence shall be observed

- o first partial DNIS digits will be received,
- o then Category will be received,
- o then rest DNIS digits,
- o then ANI digits will be received and
- o finally Category shall be received again.

Note :

- o At the most one parameter out of CDP_DNIS_DIGITS_BEFORE_ANI and CDP_DNIS_DIGITS_BEFORE_CAT should be set to a non-zero value. otherwise the behaviour of protocol shall be undetermined.
- o If non-zero, the value of this parameter should always be minimum among CDP_NUM_OF_DNIS_DIGITS (if non zero), CDP_DNIS_MaxDigits and the actual DNIS digits to be received, otherwise the behaviour of protocol shall be undetermined.
- o If this parameter is non-zero, CDP_ANI_ENABLED must be set to 1 otherwise the behaviour of protocol shall be undetermined.

*/

All INTEGER_t CDP_DNIS_DIGITS_BEFORE_ANI = 0

/*

CDP_DNIS_DIGITS_BEFORE_CAT

Inbound

This parameter determines how many DNIS Digits are to be received Before Category information,

Appendix B - Sample .sdp File

Set this parameter to 0, if Category has to be received after Complete DNIS

If set to non-zero value, following sequence shall be observed

- o first partial DNIS digits shall be received,
- o then Category will be received,
- o then rest DNIS digits,
- o then again category (If CDP_ANI_ENABLED is 1)
- o then ANI digits will be received (If CDP_ANI_ENABLED is 1)
- o finally Category shall be received again

Note :

- o At the most one parameter out of CDP_DNIS_DIGITS_BEFORE_ANI and CDP_DNIS_DIGITS_BEFORE_CAT should be set to a non-zero value. otherwise, the behaviour of protocol shall be undetermined.
- o If non-zero, the value of this parameter should always be minimum among CDP_NUM_OF_DNIS_DIGITS (if non zero), CDP_DNIS_MaxDigits and the actual DNIS digits to be received, otherwise, the behaviour of protocol shall be undetermined.

*/

All INTEGER_t CDP_DNIS_DIGITS_BEFORE_CAT = 0

/*

CDP_NUM_OF_DNIS_DIGITS

Inbound

Expected length of DNIS digits,

if 0, I-15 will terminate DNIS

This parameter should always be less than CDP_DNIS_MaxDigits

*/

All INTEGER_t CDP_NUM_OF_DNIS_DIGITS = 0

/*

CDP_NUM_OF_ANI_DIGITS

Inbound

Expected length of ANI digits,

if 0, I-15 will terminate ANI

This parameter should always be less than CDP_ANI_MaxDigits

*/

All INTEGER_t CDP_NUM_OF_ANI_DIGITS = 0

/*

CDP_DNIS_MaxDigits

CDP_ANI_MaxDigits

Inbound

Maximum Length of DNIS and ANI digits supported by the Protocol

*/

All INTEGER_t CDP_DNIS_MaxDigits = 16

All INTEGER_t CDP_ANI_MaxDigits = 16

GlobalCall™ E-1/T-1 Technology User's Guide for Linux and Windows

```
/*
*****
CDP_TimeToRecognizeAnswer
*****
This timeout is used if the inbound application
accepts the call, but instead of answering,
drops the call. In that case cas_answer is
sent across and this timer is started. After
this much time clearbwd is sent.
*/

All INTEGER_t CDP_TimeToRecognizeAnswer = 500

/*
*****
CDP_MAKECALLTIMEOUT
*****

This parameter defines the maximum timeout in milliseconds for
receiving CAS_ANSWER from remote (Inbound) Exchange after we have
switched to delivered call state. In case the timer expires call
cleared with a reason of timeout.

NOTE: This parameter is picked from the CDP file only if "Timeout"
==== specified in CMD_MakeCall is 0.
*/

All INTEGER_t CDP_MAKECALLTIMEOUT = 30000

/*
*****
*
*
*
* ADD ANY MORE USER CONFIGURABLE PAMAMETERS IF NECESSARY
*
*
*
*****
*/

/*
*****
END OF FILE
*****
*/
```

Index

- .
- .cdp file, 63
- .prm file
 - Linux, 65
 - Windows, 65
- @
- @0
 - ICAPI special parameter, 63
- A**
- address signals, 9
- alarm handling, 28
- analog links, 17
- ANI, 9
- answering machine detection, 43
- application debugging, 73
- asynchronous mode, 44
- automatic number identification, 9
- B**
- backward signal, 8
- billing rates, 49
- B-tones, 9
- C**
- cadence break, 42
- call analysis, 5
- call progress tones, 8
 - gc_SetParm(), 19
- called party, 8
- calling party, 8
- CAS
 - Channel Associated Signaling, 7
- central office, 8
- Channel Associated Signaling
 - CAS, 7
- CO
 - central office, 8
- code example
 - call progress tones, 19
- compelled signaling, 12
- configuration file
 - icapi.cfg, 73
- Connect, 18
- connect detection, 18
- country dependent parameter file, 63
- CPE
 - customer premises equipment, 8
- customer premises equipment, 8
- D**
- D4 frame, 6
- D4 superframe, 6
- DDI
 - Direct Dialing In, 13
- DDI digits, 38
- debugging applications, 73
 - ICAPI protocols, 73

dedicated voice resource
 example of, 54

destination CO, 4

dial tone, 3

dial number identification service, 13

DID
 direct inward dialing, 13

Direct Dialing In, 13

direct inward dialing, 13

direction indicator
 in protocol name, 60

DNIS
 dial number identification service,
 13

DTME
 description, 3

E

E&M
 interface, 6
 signals, 6

E-1 protocol name, 60

enhanced call progress
 for PDK protocols, 43

ESF
 extended superframe, 7

extended superframe
 ESF, 7

F

fax machine detection, 43

firmware parameter file, 63
 .prm for Linux, 65
 .prm for Windows, 65

flash-hook, 5

forward signal, 8

frequency overlap, 4

G

gc_AcceptCall(), 37

gc_AnswerCall(), 38

gc_BlindTransfer(), 39

gc_CallAck(), 38

gc_Close(), 39

gc_CompleteTransfer(), 39

gc_Detach(), 39

gc_DropCall(), 39
 PDK protocol considerations, 41

gc_Extension(), 41

gc_GetCallInfo(), 42

gc_HoldCall(), 43

gc_MakeCall(), 43
 PDK protocol considerations, 44

gc_OpenEx(), 47
 devicename parameter, 47

gc_OpenEx() devicename
 example for ICAPI protocols, 48

gc_RetrieveCall(), 49

gc_SetBilling(), 49

gc_SetChanState(), 50

gc_SetParm(), 51
 Global Tone Detection, 51

gc_SetupTransfer(), 52

gc_Start(), 51

gc_StartTrace(), 51

- gc_Stop(), 51
- gc_SwapHold(), 52
- GCEV_ALERTING event, 45
- GCEV_ANSWERED event, 45
- GCEV_BLOCKED event, 28
- GCEV_UNBLOCKED event, 28
- gcpdkrt.h
 - header file, 27
- Global Tone Detection, 15
 - gc_SetParm(), 51
- Group A backward signal, 10
- Group B backward signal, 10
- Group I forward signal, 10
- Group II forward signal., 10
- GTD, 17
 - Global Tone Detection, 15
- I**
- IC_MAKECALL_BLK structure, 45
- IC_MAKECALL_BLK structure, 45
- ICAPI protocol
 - debugging applications, 73
 - file set, 61
- icapi.cfg file
 - location of, 69
- icapi.h
 - header file, 27
- icapi.inf file
 - generation of, 70
- inbound protocol
 - support for, 59
- incoming register, 8
 - backward signals, 11

- international networks, 10
- interregister signals, 8
- L**
- line signaling, 18
- local CO, 4
- local loop, 3
- log file, 73
 - gc_Start(), 51
 - ICAPI protocols, 73
- logging
 - disabling for PDK protocols, 74
 - enabling for ICAPI protocols, 70
 - enabling for PDK protocols, 74
 - number of channels to monitor, 70
- M**
- MF
 - description, 4
- MF SOCOTEL
 - protocol name, 60
 - signaling, 12
- N**
- naming convention
 - protocol, 60
- national networks, 10
- national traffic, 9
- network resource, 27
- O**
- off-hook, 6
- operator intercept, 5
- outbound call, 8
- outbound protocol

support for, 59
outgoing register, 8

P

ParamFile file, 65
PCM carrier system, 7
PDK protocol
 file set for ICAPI, 62
PDK protocol considerations
 enhanced call progress, 43
 gc_DropCall(), 41
 gc_MakeCall() timeout, 44
PDK_MAKECALL_BLK structure, 46
predefined tones, 51
process identification number, 73
protocol
 file set for ICAPI, 61
 file set for PDK, 62
 naming convention, 60
 sample component names, 62
 service layer parameters, 31
 support, 59
 troubleshooting, 73
protocol component
 .cdp file, 63
 .prm firmware file, 63
 protocol module, 62
protocol error, 44
protocol handler, 28
protocol module
 ICAPI, 30, 62
 PDK, 30, 62
protocol name
 conventions, 61
 country code, 60
 direction indicator, 60

E&M, 60
E-1 protocol, 60
MF SOCOTEL, 60
protocol type, 60
R1 MFC protocol, 60
R2 protocol, 60
T-1 E&M with MF protocol, 60

pulse dialing, 3

R

R1 MFC protocol name, 60
R2 MF
 multifrequency combinations, 10
 signaling, 7
 signaling concepts, 8
R2 MF signaling, 8
R2 MFC protocol name, 60
R2 network, 8
R2 tones, 8
R2MF
 compelled signaling, 12
 forward signal, 10
rates
 billing, 49
resource sharing, 27
ringback, 5
ringback tone, 17
ringing tone, 5
robbed bit signaling, 6
rotary dialing, 3
S
Service layer parameters, 31
service states, 50

setting up a call, 45, 46

SF

single frequency, 6

shared voice resource

example, 56

signaling bits, 6

signaling concepts

R2MF, 8

single frequency

SF, 6

Socotel backbone, 10

supervisory signaling

R2MF, 9

T

T-1 E&M protocol name, 60

T-1 trunk, 6

tonal information

R2, 9

tone dialing, 3

tone template, 17

commenting out, 18

TONEOFF event, 18

TONEON event, 18

troubleshooting, 73

V

voice detection, 43

voice resource, 27

attaching, 28

dedicated, 27

detaching, 28

share, 56

voice resources

dedicated, 53

