

Dialogic[®] Global Call Analog Technology Guide

Copyright © 2003-2008 Dialogic Corporation

05-1041-009

Copyright and Legal Notice

Copyright © 2003-2008, Dialogic Corporation. All Rights Reserved. You may not reproduce this document in whole or in part without permission in writing from Dialogic Corporation at the address provided below.

All contents of this document are furnished for informational use only and are subject to change without notice and do not represent a commitment on the part of Dialogic Corporation or its subsidiaries ("Dialogic"). Reasonable effort is made to ensure the accuracy of the information contained in the document. However, Dialogic does not warrant the accuracy of this information and cannot accept responsibility for errors, inaccuracies or omissions that may be contained in this document.

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH DIALOGIC® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN A SIGNED AGREEMENT BETWEEN YOU AND DIALOGIC, DIALOGIC ASSUMES NO LIABILITY WHATSOEVER, AND DIALOGIC DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF DIALOGIC PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY INTELLECTUAL PROPERTY RIGHT OF A THIRD PARTY.

Dialogic products are not intended for use in medical, life saving, life sustaining, critical control or safety systems, or in nuclear facility applications.

Due to differing national regulations and approval requirements, certain Dialogic products may be suitable for use only in specific countries, and thus may not function properly in other countries. You are responsible for ensuring that your use of such products occurs only in the countries where such use is suitable. For information on specific products, contact Dialogic Corporation at the address indicated below or on the web at www.dialogic.com.

It is possible that the use or implementation of any one of the concepts, applications, or ideas described in this document, in marketing collateral produced by 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 provide any intellectual property licenses with the sale of Dialogic products other than a license to use such product in accordance with intellectual property owned or validly licensed by Dialogic and no such licenses are provided except pursuant to a signed agreement with Dialogic. More detailed information about such intellectual property is available from Dialogic's legal department at 9800 Cavendish Blvd., 5th Floor, Montreal, Quebec, Canada H4M 2V9. **Dialogic encourages all users of its products to procure all necessary intellectual property licenses required to implement any concepts or applications and does not condone or encourage any intellectual property infringement and disclaims any responsibility related thereto. These intellectual property licenses may differ from country to country and it is the responsibility of those who develop the concepts or applications to be aware of and comply with different national license requirements.**

Dialogic, Dialogic Pro, Brooktrout, Cantata, SnowShore, Eicon, Eicon Networks, Eiconcard, Diva, SIPcontrol, Diva ISDN, TruFax, Realblobs, Realcomm 100, NetAccess, Instant ISDN, TRXStream, Exnet, Exnet Connect, EXS, ExchangePlus VSE, Switchkit, N20, Powering The Service-Ready Network, Vantage, Making Innovation Thrive, Connecting People to Information, Connecting to Growth and Shiva, among others as well as related logos, are either registered trademarks or trademarks of Dialogic. Dialogic's trademarks may be used publicly only with permission from Dialogic. Such permission may only be granted by Dialogic's legal department at 9800 Cavendish Blvd., 5th Floor, Montreal, Quebec, Canada H4M 2V9. Any authorized use of Dialogic's trademarks will be subject to full respect of the trademark guidelines published by Dialogic from time to time and any use of Dialogic's trademarks requires proper acknowledgement.

Windows is a registered trademark of Microsoft Corporation in the United States and/or other countries. Other names of actual companies and products mentioned herein are the trademarks of their respective owners.

Publication Date: September 2008

Document Number: 05-1041-009

Table of Contents

| | |
|---|-----------|
| 1. How to Use This Guide | 11 |
| 1.1. Organization of This Guide..... | 11 |
| 1.2. Dialogic® Products That Support Analog Interfaces..... | 12 |
| 1.3. Related Information | 12 |
| 2. Developing Global Call Analog Loop Start Applications..... | 15 |
| 2.1. Header Files | 15 |
| 2.2. Analog Telephone Calls..... | 16 |
| 2.2.1. Inbound Analog Calls..... | 16 |
| 2.2.2. Outbound Analog Calls..... | 17 |
| 2.3. Enhanced Call Analysis Concepts | 17 |
| 2.4. Analog Signaling..... | 18 |
| 2.5. Global Tone Detection Considerations | 20 |
| 2.6. Call Progress and Call Analysis..... | 20 |
| 2.6.1. Call Analysis with Dialogic® DM3 Boards..... | 21 |
| 2.6.2. Call Analysis for PDKRT Protocols..... | 25 |
| 2.7. Loop Current Reversal Detection on Dialogic® DM3 Boards | 26 |
| 2.7.1. Enabling Reception of the GCEV_EXTENSION Event..... | 28 |
| 2.7.2. Updating the CONFIG File | 30 |
| 2.8. Analog Call Transfer Support on Dialogic® Springware Boards..... | 30 |
| 2.8.1. Call Transfer Overview | 31 |
| 2.8.2. Configuring the CDP File..... | 31 |
| 2.9. Supervised Call Transfer on Dialogic® DM3 Boards | 32 |
| 2.9.1. Basic Call Transfer Scenario | 33 |
| 2.9.2. Call Transfer APIs..... | 34 |
| 2.9.3. Application Development Notes | 35 |
| 2.9.4. PBX Testing | 35 |
| 2.9.5. PBX Integration Issues | 37 |
| 2.9.6. Configuring the Software | 37 |
| 2.10. Resource Association | 38 |
| 2.11. Alarm Handling..... | 39 |
| 2.12. Network Call Termination | 39 |
| 2.13. Run Time Configuration of the PDKRT Call Control Library | 40 |
| 2.14. Run Time Configuration of PDK Protocol Parameters..... | 41 |
| 2.15. Determining Protocol Version | 44 |
| 2.16. Programming Guidelines for PDK Analog Applications..... | 45 |

| | |
|---|-----------|
| 3. Applying Global Call Functions to Analog Loop Start Applications..... | 47 |
| 3.1. gc_AcceptCall() | 48 |
| 3.2. gc_AnswerCall() | 48 |
| 3.3. gc_Attach() and gc_AttachResource() | 48 |
| 3.4. gc_BlindTransfer() | 48 |
| 3.5. gc_Detach() | 49 |
| 3.6. gc_DropCall() | 49 |
| 3.7. gc_GetANI() | 49 |
| 3.8. gc_GetCallInfo() | 50 |
| 3.9. gc_GetParm() | 50 |
| 3.10. gc_MakeCall() | 51 |
| 3.10.1. Use of the timeout Parameter | 51 |
| 3.10.2. Other gc_MakeCall() Considerations | 51 |
| 3.10.3. PDK_MAKECALL_BLK | 52 |
| 3.11. gc_OpenEx() | 54 |
| 3.11.1. gc_OpenEx() with Dialogic® Springware Boards | 54 |
| 3.11.2. gc_OpenEx() with Dialogic® DM3 Boards | 55 |
| 3.12. gc_ReleaseCall() and gc_ReleaseCallEx() | 56 |
| 3.13. gc_ResetLineDev() | 57 |
| 3.14. gc_SetParm() | 57 |
| 3.15. gc_Start() | 58 |
| 3.16. gc_StartTrace() | 58 |
| 3.17. gc_WaitCall() | 59 |
| 4. Resource Allocation and Routing | 61 |
| 5. Analog Protocols | 63 |
| 5.1. Protocols Supported | 63 |
| 5.2. Protocol File Naming Conventions | 64 |
| 5.3. Protocol Components | 65 |
| 5.3.1. Protocol Modules | 65 |
| 5.3.2. Country Dependent Parameter (.cdp) Files | 65 |
| 6. Debug Utilities | 67 |
| 6.1. Enabling and Disabling the Logging | 67 |
| 6.2. Populating and Using a CCLIB_START_STRUCT | 68 |
| 6.3. Defining the GC_PDK_START_LOG Environment Variable | 74 |
| 6.4. Extended Logging | 75 |
| 6.4.1. gc_ExtensionFunction() | 75 |
| 6.4.2. PDK_XTEN_LOG_FUNC | 76 |
| 6.4.3. Extended Logging Code Example | 77 |

Table of Contents

Index..... 79

List of Figures

| | |
|--|----|
| Figure 1. Basic Call Transfer Scenario | 34 |
|--|----|

Dialogic® Global Call Analog Technology Guide

List of Tables

| | |
|---|----|
| Table 1. Signaling Used to Dial | 19 |
| Table 2. Global Call Call Progress Settings | 23 |
| Table 3. Call Analysis Support on Dialogic® DM3 Analog Boards | 24 |
| Table 4. Reasons for Network Call Termination | 40 |
| Table 5. Configurable PDKRT Call Control Library Parameters | 40 |
| Table 6. CDP Parameters | 41 |
| Table 7. PSL and SYS Parameters | 42 |
| Table 8. Configurable PDK Protocol Parameters | 42 |
| Table 9. Analog Call Conditions and Results | 52 |
| Table 10. PDK_MAKECALL_BLK Field Descriptions | 53 |
| Table 11. Parameters Supported, gc_GetParm() and gc_SetParm() | 58 |
| Table 12. Protocol File Naming Conventions | 64 |
| Table 13. PDK North American Analog Protocol File Set | 65 |
| Table 14. cclib_data Fields and Values | 69 |
| Table 15. Loglevel Parameter Values | 70 |
| Table 16. Service Parameter Values | 72 |
| Table 17. Cachedump Parameter Values | 73 |
| Table 18. Sample Channel Parameter Values | 73 |
| Table 19. PDK_XTEN_LOG_FUNC Field Descriptions | 76 |

Dialogic® Global Call Analog Technology Guide

1. How to Use This Guide

This guide is for users who choose to use the Dialogic® Global Call application programming interface (API) and related software to develop Linux or Windows® applications in an analog loop start interface environment.

Reference information and programming guidelines for the Dialogic® Global Call API are given in the companion documents, *Dialogic® Global Call API Library Reference* and *Dialogic® Global Call API Programming Guide*. Certain Global Call functions have additional functionality or perform differently when used in an analog loop start environment. The general function descriptions in the *Dialogic® Global Call API Library 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 analog loop start environment is contained in this *Dialogic® Global Call Analog Technology Guide*.

Throughout this document, the terms *analog environment*, *analog loop start*, and *analog interface* refer to the telephone line interface that receives analog voice and telephony signaling information from the telephone network.

NOTE: Differences between the implementation of a Global Call application in a Linux or a Windows® environment are either described parenthetically or are presented in separate paragraphs or sections. Information that is specific to the use of Dialogic® DM3 Boards or Dialogic® Springware Boards is identified explicitly. Information that is specific to the use of the PDKRT call control library is identified explicitly.

The rest of this chapter describes the organization of this guide, the products covered by this guide, and related publications.

1.1. Organization of This Guide

The information in this guide is organized as follows:

Chapter 2. Developing Global Call Analog Loop Start Applications presents guidelines for developing analog loop start applications.

Dialogic® Global Call Analog Technology Guide

Chapter 3. Applying Global Call Functions to Analog Loop Start Applications describes the additional functionality or the difference in performance of specific Global Call functions when used in an analog loop start environment.

Chapter 4. Resource Allocation and Routing describes using dedicated voice resources in an analog loop start environment.

Chapter 5. Analog Protocols describes the protocol conventions used and programming considerations when incorporating individual country protocol(s) into your application.

Chapter 6. Debug Utilities describes the diagnostic tools available for debugging a Global Call application.

1.2. Dialogic® Products That Support Analog Interfaces

The Dialogic® Global Call software provides a consistent interface across Dialogic® products interfaced to various networks (for example, E1 CAS, T1 robbed bit, E1 ISDN, T1 ISDN, analog, SS7, and IP). See the Release Guide for your Dialogic® System Release Software for the Dialogic® products that support analog interfaces.

1.3. Related Information

Use this guide in conjunction with the following manuals:

- *Dialogic® Global Call API Library Reference* – provides a reference to all functions, events, data structures, and error codes in the Dialogic® Global Call API library.
- *Dialogic® Global Call API Programming Guide* – provides guidelines for those choosing to develop applications using the Dialogic® Global Call API Library.
- *Dialogic® Global Call Country Dependent Parameters (CDP) for PDK Protocols Configuration Guide* – describes the country dependent parameters needed for utilizing Global Call.

1. How to Use This Guide

The following information is also useful:

- Release Guide for your Dialogic® System Release Software – provides information about the system release, system requirements, software and hardware features, supported hardware, and release documentation.
- Release Update for your Dialogic® System Release Software – describes compatibility issues, restrictions and limitations, known problems, and late-breaking updates or corrections to the release documentation. The Release Update is updated with new information as needed during the lifecycle of the release.
- <http://www.dialogic.com/support/default.asp> – Dialogic Technical Support Web site that contains developer support information, downloads, release documentation, technical notes, application notes, a user discussion forum, and more.

2. Developing Global Call Analog Loop Start Applications

This chapter offers information for programmers choosing to design and code Dialogic® Global Call applications in a Linux or Windows® environment. Information for developing analog loop start applications is provided. Topics include the following:

- Header files
- Analog telephone calls
- Enhanced call analysis concepts
- Analog signaling
- Global tone detection considerations
- Call progress and call analysis
- Loop current reversal detection on Dialogic® DM3 Boards
- Analog call transfer support on Dialogic® Springware Boards and Dialogic® DM3 Boards
- Resource association
- Alarm handling
- Network call termination
- Run time configuration of the PDKRT call control library
- Run time configuration of PDK protocol parameters
- Determining protocol version
- Programming guideline for PDK analog applications

2.1. Header Files

In addition to the common Global Call 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 for analog technology:

- *gcpdkrt.h* - required when using PDK error codes, the PDK_MAKECALL_BLK structure for call analysis, or logging via the **gc_Start()** function.
- *dm3cc_parm.h* - required when developing applications for Dialogic® DM3 Boards; contains the SetIDs and ParmIDs for the different technologies.

2.2. Analog Telephone Calls

For each analog loop start channel, the **gc_OpenEx()** function is used to open the voice line device and the telephone network interface device (interface to the loop start telephone line or trunk).

For Dialogic® Springware Boards, the **gc_LoadDxParm()** function is invoked to set voice parameters to be used for the voice channel associated with a line device. These voice parameters are specified in a user-created voice channel parameter (.vcp) ASCII text file. Parameters that are not specified will be assigned their default value automatically. The voice channel parameters include all channel-level parameters set by the voice function, **dx_setparm()**, and all enhanced call analysis parameters defined in the voice DX_CAP data structure. This feature is not currently available on Dialogic® DM3 Boards.

2.2.1. Inbound Analog Calls

For inbound calls, after the operations described in *Section 2.2. Analog Telephone Calls* complete, a **gc_WaitCall()** function is issued and waits for an inbound call request on the loop start network interface device.

- When the **gc_WaitCall()** function is issued synchronously, the function waits for the number of rings defined by the default number of rings parameter (set by the .cdp file—not available on Dialogic® DM3 Boards, as calls are offered to the application as soon as the firmware has all of the information needed to present the call) or for time-out to expire. When either condition occurs, the function returns.
- When the **gc_WaitCall()** function is issued asynchronously, the function completes when an unsolicited GCEV_OFFERED event occurs.

When an inbound call is received, the **gc_AnswerCall()** function establishes the conditions for answering the call, answers the call, and continues to monitor for a disconnect. The **rings** parameter of the **gc_AnswerCall()** function is not used, as the analog protocol does not generate the actual ringback; it is generated by the analog switch.

During the call, Global Call continually tests for call disconnect by monitoring for disconnect tones or for a loop current change.

2. Developing Global Call Analog Loop Start Applications

The `gc_CallAck()` function is not supported for analog calls.

2.2.2. Outbound Analog Calls

For an outbound call, after the operations described in *Section 2.2. Analog Telephone Calls* complete, the `gc_MakeCall()` function is invoked to make an outgoing call using the specified loop start network and voice resources. First, the channel used to make the outbound call is taken off-hook. Then the number is dialed using DTMF signaling, MF tone signaling, or pulse dialing. (Pulse dialing is not available on Dialogic® DM3 Boards.) Call progress tones are monitored to track the progress (current status) of the call. Enhanced call analysis is used for outbound analog telephone calls.

On Dialogic® Springware Boards, the call progress tones can be changed from their default values by using the `dx_` API to change the appropriate default tones in the firmware. (Refer to the *Dialogic® Voice API Library Reference*.) Other call analysis parameters can be set by the `gc_LoadDxParm()` function (Springware Boards only). Global Call analog technology can be configured to not use call progress tones, but the protocol will transition to the connected state immediately after dialing, as there is no way for the protocol to determine connection status with call progress analysis.

2.3. Enhanced Call Analysis Concepts

Dialogic analog call technology uses a method of signal identification for call analysis that can also detect fax machines and answering machines.

NOTE: All call analysis parameters (“basic only” and “enhanced”) are supported by Global Call analog call technology.

Call analysis is initiated when a call is dialed. Call parameters are determined by the parameters and values defined in the voice `DX_CAP` call analysis parameter data structure. On Dialogic® Springware Boards, 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. (The default values cannot be changed on Dialogic® DM3 Boards.) For a detailed description of enhanced call analysis (Perfect Call) and how to use call analysis, see the *Dialogic® Voice API Programming Guide*.

Dialogic® Global Call Analog Technology Guide

For each analog call, signaling information is sent to the local central office (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. The number of tones used and the frequency combinations used to convey this signaling information vary from country to country. Also, whenever a call is switched via networks that do not support or pass caller identification information, then this information can be lost.

2.4. Analog Signaling

Analog signaling (DTMF, MF tones, or pulses) transmit the telephone number of the called party to the local central office (CO). 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 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 explicitly identify the called party.

The destination CO checks the called party's line to determine if it is idle or busy. If the called party's line is idle, 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, the destination CO sends this information backwards to the calling party via tones.

NOTE: Analog technology does not provide a means to physically block or unblock an analog line.

Pulse dialing (also called rotary dialing) sends digit information to the CO by momentarily opening and closing (or breaking) the electrical loop from the calling party to the CO. This electrical loop is broken once for the digit 1, twice

2. Developing Global Call Analog Loop Start Applications

for 2, etc., and 10 times for the digit 0. (Pulse dialing is not available on Dialogic® DM3 Boards.)

DTMF and MF signaling use a multifrequency code system wherein each DTMF or MF signal is composed of two frequencies, as listed in *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).

Some MF digits use approximately the same frequencies as DTMF digits; for example, the digit 4 uses 770 and 1209 Hz for DTMF 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. Digit detection accuracy depends on the digit sent and the type of detection, MF or DTMF, enabled when the digit is detected. See the *Dialogic® Voice API Library Reference* for details.

Table 1. Signaling Used to Dial

| Code | Pulse (clicks) | DTMF (Hz) | MF (Hz) |
|------|----------------|-----------|------------|
| 1 | 1 | 697, 1209 | 700, 900 |
| 2 | 2 | 697, 1336 | 700, 1100 |
| 3 | 3 | 697, 1477 | 900, 1100 |
| 4 | 4 | 770, 1209 | 700, 1300 |
| 5 | 5 | 770, 1336 | 900, 1300 |
| 6 | 6 | 770, 1477 | 1100, 1300 |
| 7 | 7 | 852, 1209 | 700, 1500 |
| 8 | 8 | 852, 1336 | 900, 1500 |
| 9 | 9 | 852, 1477 | 1100, 1500 |
| 0 | 10 | 941, 1336 | 1300, 1500 |

| Code | Pulse (clicks) | DTMF (Hz) | MF (Hz) |
|-------------|-----------------------|------------------|----------------|
| * | - | 941, 1209 | 1100, 1700 |
| # | - | 941, 1477 | 1500, 1700 |

2.5. Global Tone Detection Considerations

The Dialogic® Global Call API provides network device independence by shielding the application from protocol-specific details while giving access to each protocol's full range of features.

Since global tone detection (GTD) tones are used for call analysis, the tone definitions are sent to the firmware when the **gc_OpenEx()** function is issued. The voice channel must be idle. Any pre-existing tones are deleted.

CAUTION

The application must **not** delete tones after the tones are downloaded, 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_OpenEx()** function. The tone IDs cannot be in the range from 101-189.

2.6. Call Progress and Call Analysis

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

NOTE: In Global Call terminology, the term call analysis is used interchangeably with the term call progress.

2. Developing Global Call Analog Loop Start Applications

Global Call call analysis uses global tone detection (GTD) to detect voice, fax, busy, fast busy, ringback, and Special Information Tones (SIT).

The **gc_MakeCall()** function defines the maximum time (in seconds) within which a call must be answered. Within that interval, busy and ringback tones can be detected. Global Call will disconnect an outbound call and report a GCEV_CALLSTATUS, GCEV_DISCONNECTED, or GCEV_TASKFAIL event to the application if the call is not answered within the default time-outs defined by the protocol or the **gc_MakeCall()** function. Global Call can also count the number of rings and report one of these events if the maximum number of rings is reached. The maximum number of rings can be changed by using the **gc_LoadDxParm()** function (Dialogic® Springware Boards only) to change the Global Call **ca_nbrdna** voice call analysis parameter; otherwise the default value of four rings is used. (The default value cannot be changed on Dialogic® DM3 Boards.)

The ringback tone heard on any specific call depends on the specific CO that is serving the called party, not the local CO. The ringback tone must be known in order to complete a call. The ringback tone generates a GCEV_ALERTING event, which is reported to the application.

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 *Section 3.8. gc_GetCallInfo()* for a list of the values that may be returned when the **info_id** parameter is CONNECT_TYPE.

The following sections discuss:

- Call analysis with Dialogic® DM3 Boards
- Call analysis for PDKRT protocols

2.6.1. Call Analysis with Dialogic® DM3 Boards

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

Dialogic® Global Call Analog Technology Guide

The information included below is specific to the analog technology and is provided for backward compatibility only. For new applications, it is recommended to use the cross-technology call progress analysis method described in the *Dialogic® Global Call API Programming Guide*.

There are two methods available for call analysis when using Dialogic® DM3 Boards: the Global Call method and the **dx_dial()** method.

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

After the normal **gc_MakeCall()** processing finishes and **GCEV_CONNECTED** event is sent, call analysis runs and generates a **GCEV_MEDIADETECTED** event that tells the application the result of the analysis (for example, fax, PVD, or PAMD is detected).

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

- If the call is successful, **gc_MakeCall()** finishes and a **GCEV_CONNECTED** event is sent, call analysis runs, and generates a **GCEV_MEDIADETECTED** event. The **gc_ResultValue()** and **gc_GetCallInfo()** functions can then be used to get more information about the type of media detected, such as voice, answering machine, and fax.
- If the call is not successful—for example, there is no ringback—a **GCEV_DISCONNECTED** event is generated and the **gc_ResultValue()** function can be used to retrieve the reason for the failure. See the *Dialogic® Global Call API Library Reference* for error codes and the *gcerr.h* file for more information.

NOTE: The information above applies when using **gc_MakeCall()** in asynchronous or synchronous mode. However, in synchronous mode, since the **gc_MakeCall()** function must complete, the **GCEV_MEDIADETECTED** event is generated after the call is connected.

2. Developing Global Call Analog Loop Start Applications

GCPR_MEDIADETECT and **GCPR_CALLPROGRESS** parameter settings for **gc_SetParm()** actually allow the application to specify whether pre- or post-connect call analysis or both should be activated. This method for achieving this is shown in *Table 2*.

Table 2. Global Call Call Progress Settings

| | GCPR_CALLPROGRESS=GCPV_DISABLE | GCPR_CALLPROGRESS=GCPV_ENABLE (default) |
|--|---------------------------------------|--|
| GCPR_MEDIADETECT=GCPV_DISABLE (default) | No call progress | Pre-connect call progress only |
| GCPR_MEDIADETECT=GCPV_ENABLE | No call progress | Full call progress |

As can be seen in this table, the default behavior (**GCPR_MEDIADETECT = GCPV_DISABLE**) disables media detection but actually activates pre-connect call progress for Dialogic® DM3 analog. To enable full call progress analysis, set the **GCPR_MEDIADETECT** parameter to **GCPV_ENABLE** for the respective channel.

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

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

Table 3 explains call analysis support on Dialogic® DM3 analog via the Global Call interface.

Table 3. Call Analysis Support on Dialogic® DM3 Analog Boards

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

2. Developing Global Call Analog Loop Start Applications

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

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

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

2.6.2. Call Analysis for PDKRT Protocols

NOTE: The information in this section is applicable to Dialogic® Springware Boards only. Dialogic® DM3 Boards do not use PDKRT analog protocols. On DM3 Boards, the analog protocol is embedded in the firmware.

The Dialogic® Protocol Development Kit Run-Time (PDKRT) library uses default tones defined in the Dialogic® Voice library for recognition of call progress tones. Any call progress tone defined by the Voice library will be detected. See the *Dialogic® Voice API Programming Guide* for more information about the default tones and the methods used to change the tones.

PDKRT protocols support call analysis via both the **gc_MakeCall()** function and two PSL parameters, **PSL_MakeCall_CallProgress** and **PSL_MakeCall_MediaDetect** defined in the .cdp file.

For call progress, when the **PSL_MakeCall_CallProgress** parameter is set to 0, call progress is disabled. When the **PSL_MakeCall_CallProgress** parameter is set to 1, call progress is enabled. When the **PSL_MakeCall_CallProgress**

Dialogic® Global Call Analog Technology Guide

parameter is set to 2, call progress is enabled unless NO_CALL_PROGRESS is specified in the PDK_MAKECALL_BLK structure used by the **gc_MakeCall()** function.

For media type detection, when the **PSL_MakeCall_MediaDetect** parameter is set to 1, media type detection is enabled. When the **PSL_MakeCall_MediaDetect** parameter is set to 2, media type detection is disabled unless MEDIA_TYPE_DETECT is specified in the PDK_MAKECALL_BLK structure used by the **gc_MakeCall()** function. In either case, 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.

2.7. Loop Current Reversal Detection on Dialogic® DM3 Boards

NOTE: The information in this section is applicable to Dialogic® DM3 Boards only, specifically, to the Dialogic® DM/V160-LP Media Board only. With Dialogic® Springware Boards, the Dialogic® Voice API **dx_setevtmsk()** function can be used to enable detection of loop current on, loop current off, and loop current reversal call status transition events (DM_LCOFF, DM_LCON, DM_LCREV).

Certain service providers furnish polarity reversal to subscribers to signal that the called (far end) party has answered a call (polarity reversal on seizure). Upon detection of polarity reversal, the call should transition to the CONNECTED state.

With the Dialogic® DM/V160-LP Board, the unsolicited GCEV_EXTENSION event is used to notify the application when any of the call status transitions occur:

- CC_CST_LCON - loop current on detected
- CC_CST_LCOFF - loop current off detected
- CC_CST_LCREV - loop current reversal detected

A typical sequence of events is:

2. Developing Global Call Analog Loop Start Applications

1. Pre-call – Loop current is off.
2. Application makes outbound call.
Function: **gc_MakeCall()**
Events received:
GCEV_EXTENSION(CC_CST_LCON)
GCEV_DIALING
GCEV_ALERTING
3. Called (far end) party answers the call.
Events received:
GCEV_EXTENSION(CC_CST_LCREV)
GCEV_CONNECTED
4. Near end drops the call.
Function: **gc_DropCall()**
Events received:
GCEV_EXTENSION(CC_CST_LCOFF)
GCEV_DROPCALL
5. Application frees the device for another call.
Function: **gc_ReleaseCallEx()**
Event received:
GCEV_RELEASECALL

NOTE: An extra LCON event may be seen after the first expected LCON is received but before LCREV is received. This may occur due to the polarity reversal. When a polarity reversal occurs, there is a momentary loss of loop current. If this duration is 100+ ms, this triggers an LCON event (since an LCON event is defined as a transition from no loop current to loop current). The longer the duration of no loop current, the more likely it is that this will cause the extra LCON event. Since the LCON is preceded by the LS_Net_Answer CAS signal (seen in log with TSPTTrace), this extra event is not an error, but merely the current line condition.

Also, an extra LCREV event may be seen before LCOFF. This is because the Central Office may reverse the polarity on the line, to counteract the polarity reversal done previously, before terminating the call. This sequence of events is acceptable but may not be seen in all cases. It depends on the Central Office.

2.7.1. Enabling Reception of the GCEV_EXTENSION Event

The GCEV_EXTENSION event indicates that unsolicited information is received from the network or remote end point. Information about the event is contained in the EXTENSIONEVTBLK structure, which is referenced via the extevtdatap pointer in the METAEVENT structure associated with the GCEV_EXTENSION event.

The GCEV_EXTENSION CC_CST_LCON, CC_CST_LCOFF, and CC_CST_LCREV events are disabled by default. Use the standard Global Call procedure for enabling reception of the GCEV_EXTENSION event by using **gc_util_insert_parm_val()** to build a GC_PARM_BLK, followed by **gc_SetConfigData()** to enable the event. See the *Dialogic® Global Call API Library Reference* for further information about these functions.

The following code snippets show how to enable and process the GCEV_EXTENSION event. Note that:

- EXTENSIONEVT_CALLSTATUS_TRANSITION is the bitmask to enable/disable reception of call status transition events.
- CCSET_CALLSTATUS_TRANSITION is the setID.
- CCPARM_CST_TYPE is the parmID, with values:
 - CC_CST_LCON
 - CC_CST_LCOFF
 - CC_CST_LCREV

These defines will be part of the GC_PARM_BLK structure that will be associated with the GCEV_EXTENSION event that the application receives. The application parses the GC_PARM_BLK to determine the call status transition reason.

Enable GCEV_EXTENSION Event

```
int EnableCallStatusInformation()
{
    GC_PARM_BLK pParmBlock = NULL;
    long requestID;
```

2. Developing Global Call Analog Loop Start Applications

```
int iRetCode = gc_util_insert_parm_val(&pParmBlock, CCSET_EXTENSIONEVT_MSK,
GC_ACT_ADDMSK, sizeof(long), EXTENSIONEVT_CALLSTATUS_TRANSITION);

int rc = gc_SetConfigData(GCTGT_CCLIB_CHAN,
m_DevHdl,
pParmBlock,
0,
GCUPDATE_IMMEDIATE,
&requestID,
EV_ASYNC);

if(rc != GC_SUCCESS) {
    cout << "failed to set evt mask" << endl;
    return GC_ERROR;
} else {
    cout << "gc_SetConfigData() called - Call Status Transition event reception
enabled"
<< endl;
}

gc_util_delete_parm_blk(pParmBlock);

return 0;
}
```

Process GCEV_EXTENSION Event

```
GC_PARM_BLKP gcParmBlkp = NULL;
GC_PARM_DATAP t_gcParmDatap = NULL;
EXTENSIONEVTBLK *ext_evtblkp = NULL;

ext_evtblkp = (EXTENSIONEVTBLK *)meta_event.extevtdatap;
gcParmBlkp = &ext_evtblkp->parmbkp;

cout << "Received GCEV_EXTENSION event with ExtID = " << ext_evtblkp->ext_id << endl;
while (t_gcParmDatap = gc_util_next_parm(gcParmBlkp, t_gcParmDatap))
{
    switch (t_gcParmDatap->set_ID)
    {
        case CCSET_CALLSTATUS_TRANSITION:
            switch(t_gcParmDatap->parm_ID)
            {
                case CCPARM_CST_TYPE:
                    // Determine the CST Type.
                    switch (t_gcParmDatap->value_buf)
                    {
                        case CC_CST_LCON:
                            cout << "LCON detected" << endl;
                            break;
                        case CC_CST_LCOFF:
                            cout << "LCOFF detected" << endl;
                            break;
                        case CC_CST_LCREV:
                            cout << "LCREV detected" << endl;
                            break;
                    }
            }
        }
    }
}
```

Dialogic® Global Call Analog Technology Guide

```
        }  
        break;  
    default:  
        cout << "Unknown PARM ID" << endl;  
        break;  
    }  
    break;  
default:  
    cout << "Unknown SET ID" << endl;  
    break;  
}  
}
```

2.7.2. Updating the CONFIG File

The following parameter requires configuration in the *dmv160lp.config* file in order to receive polarity reversal events:

- In the **Variant 2** section of file, change **Variant PolarityDetection** from 0 to 1.

Whenever a .config file has been modified, a new .fcd file must be generated. This procedure, which must be performed before the board is started, is described in the *Dialogic® DM3 Architecture PCI Products Configuration Guide*.

2.8. Analog Call Transfer Support on Dialogic® Springware Boards

NOTE: The information in this section is applicable to Dialogic® Springware Boards only. For information about call transfer support on Dialogic® DM3 Boards, see *Section 2.9. Supervised Call Transfer on Dialogic® DM3 Boards*.

Blind and supervised analog call transfers using the Dialogic® Global Call API are supported on Dialogic® Springware Boards.

Support for analog call transfer is applicable only to standard POTS (“plain old telephone service”) line products. Proprietary private branch exchanges (PBXs) or key telephone systems (KTSs) and their related boards, such as the Dialogic® D/42JCT-U and D/82JCT-U PBX Integration Boards, are excluded from this feature since PBXs and KTSs may provide proprietary protocols for call transfers.

2. Developing Global Call Analog Loop Start Applications

2.8.1. Call Transfer Overview

There are two types of call transfers:

- **Supervised transfers** – The person transferring the call stays on the line, announces the call, and consults with the party to whom the call is being transferred before the transfer is completed.
- **Blind transfers** – The call is sent without any consultation or announcement by the person transferring the call. Blind transfers are also known as one-step or unsupervised transfers.

Supervised transfers use the following Dialogic® Global Call API functions:

- **gc_SetupTransfer()** - initiates a supervised transfer
- **gc_CompleteTransfer()** - completes a supervised transfer
- **gc_SwapHold()** – switches between the consultation call and the call pending transfer

Blind transfers use the following Dialogic® Global Call API function:

- **gc_BlindTransfer()** - initiates and completes an unsupervised (one-step) transfer

For further information about call transfers, see the Call State Models chapter of the *Dialogic® Global Call API Programming Guide*.

2.8.2. Configuring the CDP File

When using Global Call with Dialogic® Springware analog boards, the boards must be using the North American Analog Bidirectional PDK protocol, `pdk_na_an_io`, provided with the Global Call Protocols package. The Global Call Protocols package is one of the features that can be selected when installing the Dialogic® System Release Software. For information about using PDK protocols, see the *Dialogic® Global Call Country Dependent Parameters (CDP) for PDK Protocols Configuration Guide*.

Dialogic® Global Call Analog Technology Guide

To support blind and supervised call transfers, the country dependent parameters file for the North American Analog Bidirectional PDK protocol, *pdk_na_an_io.cdp*, has the following parameters that can be set by the user:

- **CDP_BtStartTimer** - For a **supervised** transfer, specifies the maximum time that the protocol will wait after issuing hookflash as a part of **gc_SetupTransfer()** and before the application issues **gc_MakeCall()**.

For a **blind transfer**, specifies the maximum time that the protocol will wait after issuing hookflash as a part of **gc_BlindTransfer()** and before the protocol completes the digit dial. Since the call is made within Global Call, this parameter can be used as a bail-out timer to dial tone detection when **CDP_Detect_DialTone** (Outbound) is enabled and none is detected during the elapsed time.

Values: Time in milliseconds. Default is 8000 (8 seconds).

- **CDP_BlindXferTime** - Specifies the delay time between the third party ringing and the controller going on-hook, i.e., disconnecting; it can be used to guard against network latencies, ensuring that the end-to-end audio path has been established before transfer.

Values: Time in milliseconds. Default is 2000 (2 seconds).

2.9. Supervised Call Transfer on Dialogic® DM3 Boards

NOTE: The information in this section is applicable to Dialogic® DM3 Boards only, specifically, to the Dialogic® DM/V160-LP Media Board only for supervised call transfers. (Blind transfers using the **gc_BlindTransfer()** function are also supported on DM3 Boards.) For information about call transfer support on Dialogic® Springware Boards, see *Section 2.8. Analog Call Transfer Support on Dialogic® Springware Boards*.

Supervised call transfer is a feature that enables a controller (party A) already in a call with another party (party B) to transfer the call to a third party (party C). The end result is a call between party B and party C. This feature is a common requirement in IVR and voicemail applications.

2. Developing Global Call Analog Loop Start Applications

2.9.1. Basic Call Transfer Scenario

The sequence of events in a supervised call transfer scenario is described below. It is assumed that party A and party B are already in a call.

1. Party A hookflashes party B, placing the call with party B on hold. This call is referred to as the “held” call.
2. Party A dials party C and waits for an answer.
3. Party A notifies party C that the transfer is about to take place. This call is referred to as the “consultation” call.
4. Optionally, party A hookflashes party C and notifies party B of the transfer.
5. Party A hangs up.
6. Parties B and C are connected and the transfer is completed.

The sequence is shown diagrammatically in the following figure.

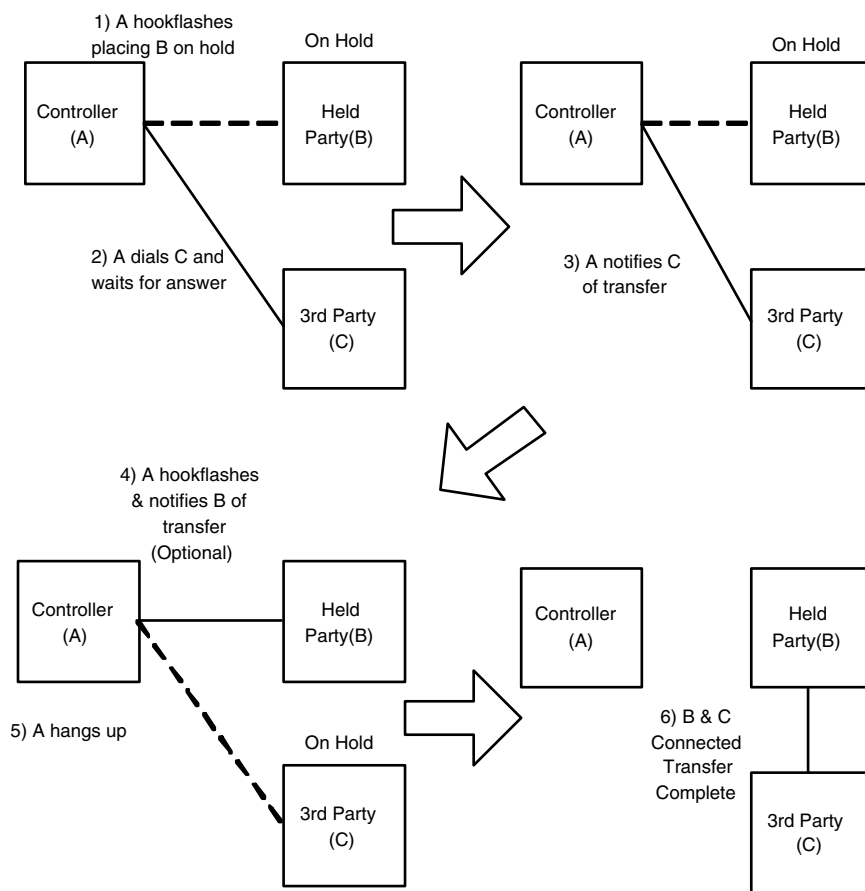


Figure 1. Basic Call Transfer Scenario

2.9.2. Call Transfer APIs

The supervised call transfer feature is provided by the following Dialogic® Global Call API functions:

- **gc_SetupTransfer()** - initiates a supervised call transfer and allocates a CRN for the consultation call
- **gc_MakeCall()** - used to make the consultation call

2. Developing Global Call Analog Loop Start Applications

- **gc_CompleteTransfer()** - used to complete the transfer and communicate to the CPE/CO equipment to connect the talk paths of the held call and the consultation call
- **gc_SwapHold()** - communicates to the CPE/CO equipment that the talk path to the controller should be “swapped” from the held call to the consultation call. This allows the controller to swap between the held party and the third party prior to the transfer. Once this API is completed, the roles of the held and consultation call are reversed.

NOTE: Depending on the PBX type and configuration, it may not be possible to use the **gc_SwapHold()** function to swap between the held call and the consultation call. For non-US protocols, the **gc_SwapHold()** function can operate correctly if the behavior of the protocol is similar to that of a US counterpart.

2.9.3. Application Development Notes

The following application development notes apply:

- When any of the parties involved in a transfer are dropped or remotely disconnected prior to calling **gc_CompleteTransfer()**, all active calls (both consultation and held calls) must be dropped using **gc_DropCall()** and the CRNs must be released using **gc_ReleaseCallEx()**.
- The **gc_ResetLineDevice()** function can be used to reset a channel and terminate all active calls when a transfer call scenario is active.
- When setting up a supervised call transfer, after the **gc_SetupTransfer()** function is issued to obtain a CRN for the consultation call, a permanent signal timer (8 seconds) starts. If the consultation call is not made within the 8-second period, the timer expires and the application receives a **GCEV_DISCONNECTED** event.

2.9.4. PBX Testing

NOTE: The call transfer feature has been tested on PBX systems that have been configured to use US protocols only.

Dialogic® Global Call Analog Technology Guide

The basic call transfer scenario as described above has been tested on the following PBX systems:

- Siemens HiCom 150E Office Pro
- Mitel SX 200
- Ericsson MD110
- Alcatel Omni PCX 4400
- Panasonic Easa-Phone KX T30810
- NEC 2400

For the Siemens HiCom 150E, the following variations in the basic call transfer scenario have also been tested:

- The controller drops the consultation call before dialing is started
 - Party B calls party A
 - Party A hookflashes and then drops the call
 - Verify: Party B is connected back to Party A
- Blind transfer
 - Party B calls party A
 - Party A hookflashes (places call with party B on hold)
 - Party A dials party C, then hangs up
 - Verify: Party B and party C connected
- The held call is dropped by Party B
 - Party B calls party A
 - Party A hookflashes (places call with party B on hold)
 - Party A calls party C (consultation call)
 - Party C picks up
 - Party B hangs up
 - Party C hookflashes
 - Verify: Party C and party A connected
- The consultation call is dropped by Party C
 - Party B calls party A
 - Party A hookflashes
 - Party B calls party C
 - Party C hangs up
 - Verify: Party B and Party A are connected again

2. Developing Global Call Analog Loop Start Applications

2.9.5. PBX Integration Issues

From a PBX perspective, call transfer is most often a sub-feature of Multi-Way Calling (MWC). MWC provides for several variations of conference call capability. Conference features are usually accessed via a flash hook followed by the dialing of an access code. The variation in the behavior of conference features needs to be taken into account when integrating a CT application on a PBX.

The following behavior needs to be considered:

- Swapping between held and consultation calls - If the PBX has conference capability enabled, issuing a second **gc_SwapHold()** could cause a three-way call to be created. This call scenario can no longer be considered a call transfer scenario.
- Remote party drop of consultation call - There are a number of possible behaviors in this scenario. These include:
 - getting disconnect treatment on party C, or
 - automatically having the talk path connected back to party A (the held call)
- Initiating a call transfer (using **gc_SetupTransfer()**), then releasing the consultation call prior to issuing **gc_MakeCall()** - Various types of “ring-back” treatments can be applied by the PBX. A “ring-back” treatment occurs when the PBX generates the ring voltage on any of the parties involved in the transfer (or conference). The duration of the generated ring can be from 1 ring (approximately 6 seconds) to 6 rings (approximately 36 seconds).

2.9.6. Configuring the Software

The following configuration instructions apply when using the Dialogic® DM/V160-LP Board with PBX systems:

- Updating the .config file
- Detecting and learning call progress tones

Updating the .config File

The following parameters require configuration in the Dialogic® DM/V160-LP Board .config file:

- **Tone_SigId4** (Disconnect Tone Supervision) must be set to a value of 238113 (a fixed tone ID) to enable disconnect tone supervision. The default value is 0x0 (disabled).
- **BtStartTimeout** (Permanent Signal Planning) must be set to a value appropriate for the PBX system being used. The default is 8000 (8 seconds). This value may need to be changed if the PBX system has a shorter time-out prior to the start of the consultation call.

Whenever a .config file has been modified, a new .fcd file must be generated. This procedure, which must be performed before the board is started, is described in the *Dialogic® DM3 Architecture PCI Products Configuration Guide*.

Detecting and Learning Call Progress Tones

On Linux, use the Dialogic® Learn Mode API and Tone Set File API to detect and learn call progress tones. For more information, see the *Dialogic® Learn Mode and Tone Set File API Software Reference*.

On Windows®, use the Dialogic® PBXpert utility to detect and learn call progress tones.

2.10. Resource Association

For Dialogic® Springware Voice Boards with on-board analog loop start devices (for example, Dialogic® D/41ESC, D/160SC-LS), a voice device and an analog loop start device comprise a single channel. Although these devices can be addressed separately, all analog signaling is processed by the associated voice device; analog signaling (ring detection and loop current detection) events are not transmitted over the SCbus. In resource sharing applications using the voice resources of a Voice Board with on-board analog loop start devices, the analog loop start device associated with a shared voice resource is disabled. See *Chapter 4. Resource Allocation and Routing* for more information.

2. Developing Global Call Analog Loop Start Applications

The Global Call line device ID (LDID) is a single ID that represents the combination of the voice resource and analog loop start (or digital) interface resource that work together to establish and to tear-down calls.

Diallogic[®] DM3 analog boards are comprised of separate voice devices and analog loop start devices, much like a digital board. As such, these devices are treated separately, with no inherent association between them. Analog loop start devices are denoted as dti devices, just like network time slots are on digital boards. When doing **gc_OpenEx()** on a Diallogic[®] DM3 analog device, it is necessary to supply the analog loop start device name as well as a voice device name, or attach a voice device to the analog loop start device after **gc_OpenEx()**.

2.11. Alarm Handling

As described in the *Diallogic[®] Global Call API Library Reference*, the GCEV_BLOCKED event indicates that a line is blocked and the application cannot issue call-related function calls. The GCEV_UNBLOCKED event indicates that the line has become unblocked.

The portion of the Global Call call control library that manages alarms, called the Global Call Alarm Management System (GCAMS), is not used. As a result, Global Call applications cannot configure alarm properties and characteristics or receive GCEV_ALARM events.

2.12. Network Call Termination

When a call is terminated by the network, an unsolicited GCEV_DISCONNECTED event is sent to the application. For analog calls, this disconnection may be due to the reasons described in *Table 4. Reasons for Network Call Termination*.

Table 4. Reasons for Network Call Termination

| Reason/Message | Global Call Result Value |
|-----------------------------------|--------------------------|
| Disconnect by loop current change | GCRV_NORMAL |
| Disconnect by tone | GCRV_NORMAL |

The application can retrieve the reason for the disconnection using the `gc_ResultInfo()` function.

2.13. Run Time Configuration of the PDKRT Call Control Library

NOTE: The information in this section is applicable to Dialogic® Springware Boards only. Dialogic® DM3 Boards do not use PDKRT analog protocols. On Dialogic® DM3 Boards, the analog protocol is embedded in the firmware.

Table 5. Configurable PDKRT Call Control Library Parameters shows the parameters of the PDKRT call control library that can be configured using the real time configuration management (RTCM) functions. The `gc_GetConfigData()` function can be used to retrieve the target object configuration, and the `gc_SetConfigData()` function can be used to update the target object configuration.

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

Table 5. Configurable PDKRT Call Control Library Parameters

| Set ID | Parm ID | Target Object Type | Description | Data Type | Access Attribute* |
|----------------|--------------|--------------------|--|-----------|-------------------|
| GCSET_CALLINFO | CONNECT_TYPE | GCTGT_CCLIB_CRN | Connect type (alternative to <code>gc_GetCallInfo()</code>) | char | GC_R_O |

***Note:** GC_R_O - retrieve only

2. Developing Global Call Analog Loop Start Applications

2.14. Run Time Configuration of PDK Protocol Parameters

NOTE: The information in this section is applicable to Dialogic® Springware Boards only. Dialogic® DM3 Boards do not use PDKRT analog protocols. On Dialogic® DM3 Boards, the analog protocol is embedded in the firmware.

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 allowed to be changed only when the channel object does not have an active call.

PSI variable parameters are interpreted by the PDK run-time component (PDKRT). The names of the PSI variable parameters (beginning with CDP_) are found in the .cdp file. The PSI parameters that can be accessed via `gc_GetConfigData()`, `gc_SetConfigData()`, and `gc_QueryConfigData()` are shown in *Table 6*.

Table 6. CDP Parameters

| Parameter Name | Data Type |
|---|-----------|
| CDP_ConnectOnNoRingBack | boolean |
| CDP_Working_Under_PBX_Env | boolean |
| CDP_Time_Before_Blind_Dialing_Under_PBX_Env | integer |
| CDP_Dgts_For_Outside_Line_In_PBX_Env | string |
| CDP_PBX_DialToneTimeout | integer |
| CDP_DialTone_As_Disconnect_In_Connected | boolean |

The PSL variable parameters are not available to the protocol state machine, but rather are used by the protocol services layer to control the behavior of various network and voice functions. The names of the PSL variable parameters begin

with PSL_ and SYS_. No variation in the names is allowed. These parameters are required to control protocol parameters (e.g., timing) or they may control the behavior of the underlying implementation. In the latter case, the parameters will most likely have a platform tag. All of these parameter names must begin with PSL. The PSL parameters that can be accessed via **gc_GetConfigData()**, **gc_SetConfigData()**, and **gc_QueryConfigData()** are shown in *Table 7*.

Table 7. PSL and SYS Parameters

| PSL Variable Name | Data Type |
|------------------------------------|------------------|
| PSL_MakeCall_CallProgress | integer |
| PSL_MakeCall_MediaDetect | integer |
| PSL_DefaultMakeCallTimeout | integer |
| PSL_ANALOG_NUM_RINGS_BEFORE_RINGON | integer |
| SYS_PSINAME | string |

Table 8 shows the Set ID and Parm ID for these parameter types.

Table 8. 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 |
| *Indicates that CAS pattern signals and tones cannot be accessed. ** GC_W_N - update only at null state | | | | |

2. Developing Global Call Analog Loop Start Applications

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

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

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

- (GCTGT_PROTOCOL_SYSTEM, Global Call protocol ID)
- (GCTGT_PROTOCOL_CHAN, Global Call line device ID)

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

The following example shows how to set the CDP_ConnectOnNoRingBack 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[25] = "CDP_ConnectOnNoRingBack";
long request_id;
LINEDEV ldev;
GC_PARM_BLK * t_pParmBlk = NULL;

/* first find the parameter info by calling gc_QueryConfigData() function */
```

Dialogic® Global Call Analog Technology Guide

```
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_ConnectOnNoRingBack" */
gc_SetConfigData(GCTGT_PROTOCOL_CHAN, ldev, t_pParmBlk, 0,
                GCUUPDATE_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);
```

2.15. Determining Protocol Version

NOTE: The information in this section is applicable to Dialogic® Springware Boards only. Dialogic® DM3 Boards do not use PDKRT analog protocols. On Dialogic® DM3 Boards, the analog protocol is embedded in the firmware.

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

```
#include <gclib.h>
#include <gcerr.h>
#include <srllib.h>
int main()
{
    LINEDEV    ldev;
    GC_PARM    parm;
    int        retcode;
    METAEVENT  metaevent;
    parm.paddress = NULL;

    int mode;
    #ifdef WIN32
    mode = SR_STASYNC|SR_POLLMODE;
    #else
    mode = SR_POLLMODE;
    #endif

    if (sr_setparm(SRL_DEVICE, SR_MODELTYPE, &mode) == -1)
    {
        // Error processing
    }
    gc_Start(NULL);
    retcode = gc_Open(&ldev, "P_pdk_na_an_io:V_dxxxB1C1", 0);
    if (retcode != GC_SUCCESS)
    {
        // Error processing
    }
}
```

2. Developing Global Call Analog Loop Start Applications

```
sr_waitevt(50);
retcode = gc_GetMetaEvent(&metaevent);
if (retcode != GC_SUCCESS)
{
    // Error processing
}
if (metaevent.flags & GCME_GC_EVENT)
{
    if (metaevent.evtttype == GCEV_UNBLOCKED)
    {
        if (gc_GetParm(ldev, GCPR_PROTVER, &parm) ==
            GC_SUCCESS)
        {
            printf("The protocol version: %s\n", parm.paddress);
        }
        else
        {
            // Error processing
            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);
gc_Stop();
return(0);
}
```

2.16. Programming Guidelines for PDK Analog Applications

- Because of a limitation in the **dx_** (voice) library, an application using the PDK analog protocol should not call any **dx_** function for a device that is expecting a Global Call termination. For example, when an application calls **gc_OpenEx()** to open a PDK analog device, the application should not call any **dx_** function on that device before receiving the GCEV_UNBLOCKED event.

Dialogic® Global Call Analog Technology Guide

- When using the **dx_setevtmsk()** function, the following **mask** settings **must** be specified for analog channels, in addition to whatever other mask settings are needed in the application:

DM_LCOFF | DM_LCON | DM_LCREV | DM_RINGS | DM_RNGOFF | DM_WINK

This is because the PDK analog library uses **dx_setevtmsk()** internally, and needs those masks in order to function correctly. Since every call to **dx_setevtmsk()** overrides the settings in the previous call, these settings must be included in all calls to **dx_setevtmsk()** on analog channels, or else the protocol will stop working.

3. Applying Global Call Functions to Analog Loop Start Applications

Certain Dialogic® Global Call functions have additional functionality or perform differently when used in an analog loop start environment. The general function descriptions in the *Dialogic® Global Call API Library Reference* do not contain detailed information on a particular technology. Information in terms of the additional functionality or the differences in performance of those functions in an analog loop start environment is contained in this chapter. Note that this information must be used in conjunction with the information presented in the *Dialogic® Global Call API Library Reference*.

The following Global Call analog loop start functions are described in this chapter:

- **gc_AcceptCall()**
- **gc_AnswerCall()**
- **gc_Attach()** and **gc_AttachResource()**
- **gc_BlindTransfer()**
- **gc_Detach()**
- **gc_DropCall()**
- **gc_GetANI()**
- **gc_GetCallInfo()**
- **gc_GetParm()**
- **gc_MakeCall()**
- **gc_OpenEx()**
- **gc_ReleaseCall()** and **gc_ReleaseCallEx()**
- **gc_ResetLineDev()**
- **gc_SetParm()**
- **gc_Start()**
- **gc_StartTrace()**
- **gc_WaitCall()**

See the *Dialogic® Global Call API Library Reference* for a listing of Global Call functions and function descriptions.

3.1. gc_AcceptCall()

In the analog protocol, the **rings** parameter is ignored.

The **gc_AcceptCall()** function provides compatibility only with other Global Call libraries and applications. If your application uses the **gc_AcceptCall()** function in an analog technology application, calling the function causes an immediate transition to the Accepted state.

The **gc_AcceptCall()** function is not applicable if using the Dialogic® DM/V160-LP-HIZ Board.

3.2. gc_AnswerCall()

The **gc_AnswerCall()** function indicates to the remote end that the connection is established (call has been answered). For analog calls, the **rings** parameter of the **gc_AnswerCall()** function is not used since this value is set by the default number of rings parameter in the .cdp file.

Set the **rings** parameter of the **gc_AnswerCall()** function to 0 for analog calls.

3.3. gc_Attach() and gc_AttachResource()

NOTE: The **gc_Attach()** function is deprecated; the preferred equivalent is **gc_AttachResource()**.

When using Dialogic® DM3 analog boards, if the **gc_Attach()** or **gc_AttachResource()** function is issued on a device in the Idle state (for example, between calls), the attach operation is performed, but also an “on-hook” transition occurs in the firmware. This transition is required to allow inbound calls to be answered.

3.4. gc_BlindTransfer()

When used with Dialogic® DM3 analog boards, the **timeout** parameter value of the **gc_BlindTransfer()** function is ignored.

3. Applying® Global Call Functions to Analog Loop Start Applications

The **gc_BlindTransfer()** function is not applicable when using the Dialogic® DM/V160-LP-HIZ Board.

3.5. gc_Detach()

When using Dialogic® DM3 analog boards, if the **gc_Detach()** function is issued on a device in the Idle state (for example, between calls), the detach operation is performed, but also an “off-hook” transition occurs in the firmware. This transition is required to ensure that no inbound calls will be processed.

3.6. gc_DropCall()

The **cause** parameter value of the **gc_DropCall()** function is ignored.

CAUTION

Before issuing a **gc_DropCall()** function, you must first terminate any voice-related function currently in progress. For example, if the play or the record function is in progress, then before you drop the call, issue a stop channel function on that voice channel and then call the **gc_DropCall()** function to drop the call.

The **gc_DropCall()** function is not applicable when using the Dialogic® DM/V160-LP-HIZ Board.

3.7. gc_GetANI()

NOTE: The **gc_GetANI()** function is deprecated in this software release. The suggested equivalent is **gc_GetCallInfo()**.

The **gc_GetANI()** function only returns the calling party’s telephone number (Directory Number, DN). Other information, such as time of day, date, and caller name, may also be available. The **gc_GetCallInfo()** function can be used to obtain this other information.

Dialogic® Global Call Analog Technology Guide

The transmission of caller ID information by the CO is protocol dependent. See your protocol in the *Dialogic® Global Call Country Dependent Parameters (CDP) for PDK Protocols Configuration Guide* for required parameter settings.

3.8. gc_GetCallInfo()

The **gc_GetCallInfo()** function can be used to retrieve ANI information such as time of day, date, and caller name, if available, from the network. When using this function to retrieve information for an inbound call, the following limitations apply to the **info_id** parameter:

- CALLTIME must be **exactly** AN_MAXCALLTIME bytes in length
- CALLNAME must be **exactly** AN_MAXCALLNAME bytes in length

When using this function to retrieve information for an outbound call, the **info_id** parameter CONNECT_TYPE contains the type of connection as returned by the function. These connection types are:

- GCCT_CAD - connection due to cadence break
- GCCT_LPC - connection due to change in loop current
- 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

3.9. gc_GetParm()

The **gc_GetParm()** function retrieves the value of the specified parameter for a line device. In addition to the GCPR_CALLINGPARTY parameter, which is common across all technologies and documented in the *Dialogic® Global Call API Library Reference*, the following parameters are supported:

- On Dialogic® DM3 Boards:
 - GCPR_CALLPROGRESS
 - GCPR_MEDIADETECT
- On Dialogic® Springware Boards:
 - GCPR_CALLPROGRESS

3. Applying Global Call Functions to Analog Loop Start Applications

See *Section 3.14. gc_SetParm()* for more information on the meaning of these parameters.

3.10. gc_MakeCall()

The **gc_MakeCall()** function is not applicable when using the Dialogic® DM/V160-LP-HIZ Board.

3.10.1. Use of the timeout Parameter

When using voice line devices, the **timeout** argument in the **gc_MakeCall()** function is supported in both the synchronous and asynchronous programming modes.

If the **timeout** value expires 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 *Dialogic® Global Call API Library Reference*. Also, see the *Dialogic® Global Call Country Dependent Parameters (CDP) for PDK Protocols Configuration Guide* for other time-outs that may apply to your analog protocol.

If all time-out values are set to 0, no time-out condition will apply.

3.10.2. Other gc_MakeCall() Considerations

For analog calls, the dialing mode can be changed by the application by including one of the following case-sensitive dialing codes in the dialing string specified by the **numberstr** parameter:

- P - for pulse mode dialing
- T - for DTMF tone mode dialing
- M - for MF tone mode dialing

When included in the dialing string, the dialing code overrides the default set by the dialing mode parameter in the .cdp file (Dialogic® Springware Boards only). (On Dialogic® DM3 Boards, inclusion of these dialing codes causes the digits **not** to be dialed.)

Dialogic® Global Call Analog Technology Guide

The **gc_MakeCall()** function description in the *Dialogic® Global Call API Library Reference* provides a table describing call conditions and results. In addition to the information in that table, the values described in *Table 9* apply when running analog technology.

Table 9. Analog Call Conditions and Results

| Condition | Event/Return Value | Result/Error Value |
|-----------------------------|--|---|
| No ringback detected | Async: GCEV_CALLSTATUS or GCEV_DISCONNECTED Sync: 0 | Async: GCRV_NORB result value Sync: EGC_NORB error |
| Operator intercept detected | Async: GCEV_CALLSTATUS or GCEV_DISCONNECTED Sync: 0 | Async: GCRV_CEPT result value Sync: EGC_CEPT error |
| Call progress stopped | Async: GCEV_CALLSTATUS or GCEV_DISCONNECTED Sync: 0 | Async: GCRV_STOPD result value Sync: EGC_STOPD error |
| SIT detection error | Async: GCEV_CALLSTATUS or GCEV_DISCONNECTED Sync: 0 | Async: GCRV_CPERROR result value Sync: EGC_CPERROR error |
| No dial tone detected | Async: GCEV_DISCONNECTED or GCEV_TASKFAIL Sync: <0 | Async: GCRV_DIALTONE Sync: EGC_DIALTONE |

3.10.3. PDK_MAKECALL_BLK

For Dialogic® Springware Boards, the PDK_MAKECALL_BLK structure contains information used by the **gc_MakeCall()** function when setting up a call. When the **gc_MakeCall()** function sets up a call, the default is to enable call analysis (call progress). This default can be changed on a call basis by setting the flags parameter in the PDK_MAKECALL_BLK data structure.

3. Applying Global Call Functions to Analog Loop Start Applications

NOTE: Control of call progress and media detection at `gc_MakeCall()` time works only when the following parameters in the `.cdp` file are set to allow application control:

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

/* Set 1 to enable, 2 to allow app control */
All INTEGER_t PSL_MakeCall_MediaDetect = 2
```

The `PDK_MAKECALL_BLK` structure is defined as follows. See *Table 10* for field descriptions.

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

Table 10. PDK_MAKECALL_BLK Field Descriptions

| Field | Description |
|------------|--|
| flags | Contains a bitmask that controls call analysis and media type detection on a per call basis. The possible values that can be ORed are: <ul style="list-style-type: none"> • <code>NO_CALL_PROGRESS</code> - To disable call analysis. • <code>MEDIA_TYPE_DETECT</code> - To enable media type detection. |
| *v_rfu_ptr | Reserved for future use. |
| ul_rfu[4] | Reserved for future use. |

Following are some examples:

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

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

Dialogic® Global Call Analog Technology Guide

```
m_gcMakecallBlk.cclib = &m_pdkMakecallBlk;
}

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

3.11. gc_OpenEx()

The **gc_OpenEx()** function opens voice channels, voice devices, or analog loop start interfaces. The following sections describe using **gc_OpenEx()** with Dialogic® Springware Boards and with Dialogic® DM3 Boards.

3.11.1. gc_OpenEx() with Dialogic® Springware Boards

For Dialogic® Springware Boards that host both voice devices and analog loop start interface devices, both the voice device and its associated analog loop start interface device are opened as a single channel. A single line device ID (LDID) identifies both the voice channel and the analog loop start interface.

A voice channel, voice device, or analog loop start interface device is specified by the **devicename** parameter using a format that includes the following information:

:P_<protocol_name>:V_<voice_channel_name>

where:

- <**protocol_name**> specifies the analog loop start protocol. Use the root file name of the analog protocol file (for example, *pdk_na_an_io*) for your country or telephone network.
- <**voice_channel_name**> specifies the name of the voice channel, voice device, or analog loop start interface device to be associated with the device being opened. Use the following format for the voice device:

dxxxB<virtual board number>C<channel or device number>

3. Applying Global Call Functions to Analog Loop Start Applications

The prefixes (P_ and V_) in **devicename** are used for parsing purposes. The order of input of these parameters may be set by the application. The fields within the **devicename** parameter must each begin with a colon.

The following example illustrates the format for defining the **devicename** parameter for voice and analog loop start interface devices when processing analog calls.

To open voice channel 2 on a Dialogic® D/160SC-LS Board identified as virtual board 3:

```
:P_pdk_na_an_io:V_dxxxB3C2
```

Global Call automatically opens both voice device 2 and analog loop start interface device 2 on virtual board 3 and internally attaches the voice device to the analog loop start interface.

NOTE: When using analog protocol, opening a board device, that is, using only the board number in the **devicename** parameter (for example, :V_dxxxB1) is not supported.

3.11.2. gc_OpenEx() with Dialogic® DM3 Boards

On Dialogic® DM3 analog boards, the voice devices and the analog loop start devices are separate devices. When calling **gc_OpenEx()**, the application can either open only the analog loop start device, or open both the analog loop start interface device and the voice device at the same time. The device(s) is specified via the **devicename** parameter using a format that includes the following information:

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

where:

- **<protocol_name>** can be any string. :P_ must be present, but the name of the protocol does not matter, as the protocol is embedded in the firmware and cannot be changed.

Dialogic® Global Call Analog Technology Guide

- **<loop_start_device_name>** specifies the name of the analog loop start device to be associated with the device being opened. Use the following format for the loop start device:

dtiB<virtual board number>T<channel or device number>

- **<voice_channel_name>** specifies the name of the optional voice channel to be associated with the device being opened. Use the following format for the voice channel name:

dxxxB<virtual board number>C<channel or device number>

The prefixes (P_, N_, and V_) in **devicename** are used for parsing purposes. The order of input of these parameters may be set by the application. The fields within the **devicename** parameter must each begin with a colon.

The following example illustrates the format for defining the **devicename** parameter for voice and analog loop start interface devices when processing analog calls.

To open loop start device 2 on a Dialogic® DM/V160 Board identified as virtual board 1 along with voice device 1 on virtual board 2:

```
:P_dm3an:N_dtiB1T2:V_dxxxB2C1
```

Global Call automatically opens both the analog loop start interface and voice device and internally attaches the voice device to the analog loop start interface.

The voice device could be opened later using the **dx_open()** function and then attached to the Global Call device using the **gc_AttachResource()** function. The application would have to manually route the resources together. Without a voice device attached to the Global Call device, no outbound calls can be made as there will be no resources to dial digits.

3.12. gc_ReleaseCall() and gc_ReleaseCallEx()

NOTE: The **gc_ReleaseCall()** function is deprecated; the preferred equivalent is **gc_ReleaseCallEx()**.

3. Applying Global Call Functions to Analog Loop Start Applications

The **gc_ReleaseCallEx()** function must be called after a **gc_DropCall()** function completes. If a new inbound call has arrived since the last **gc_DropCall()** function was issued, that call will be pending until the **gc_ReleaseCallEx()** function is called.

If a **gc_WaitCall()** function is issued asynchronously, the inbound call notification can be received immediately after the **gc_ReleaseCallEx()** function is called. If a **gc_WaitCall()** function is issued synchronously and a **gc_ReleaseCallEx()** function is issued subsequently, the inbound call will be pending until the **gc_WaitCall()** function is issued again.

The **gc_ReleaseCall()** and **gc_ReleaseCallEx()** functions are not applicable when using the Dialogic® DM/V160-LP-HIZ board.

3.13. gc_ResetLineDev()

The **gc_ResetLineDev()** function is used to ensure that voice channels are set on-hook. The **gc_ResetLineDev()** function also sets the Global Call call state to Idle. Placing each voice channel on-hook ensures that any active calls are disconnected and eliminates the possibility of leaving a line in a ringing condition.

The **gc_ResetLineDev()** function can be called only in the asynchronous mode. You must wait until the GCEV_RESETLINEDEV event is received from each voice channel before continuing to ensure that the voice channels have been set on-hook.

3.14. gc_SetParm()

The **gc_SetParm()** function sets the default parameters and all channel information associated with the specific line device. In addition to the GCPR_CALLINGPARTY parameter, which is common across all technologies and documented in the *Dialogic® Global Call API Library Reference*, the parameters listed in *Table 11* are supported.

Table 11. Parameters Supported, gc_GetParm() and gc_SetParm()

| Parameter | Level | Description | Supported on |
|-----------------------|---------|---|--|
| GCPR_CALL PROGRESS | channel | Enables or disables call progress; enabled by default. If this parameter is disabled, post-connect call progress is also disabled, regardless of the setting of GCPR_MEDIADETECT. | Dialogic® DM3 and Springware Boards |
| GCPR_MEDI ADETECT | channel | Enables or disables post-connect call progress or media detection; disabled by default. | Dialogic® DM3 Boards |

For further information about the use of the GCPR_CALLPROGRESS and GCPR_MEDIADETECT parameters, see *Section 2.6.1. Call Analysis with Dialogic® DM3 Boards*.

NOTE: The `gc_SetParm()` function is not supported when using PDK analog.

3.15. gc_Start()

For PDK protocols, the `gc_Start()` function is used to access the error and debug logging capabilities of the Dialogic® PDKRT call control library. See *Chapter 6. Debug Utilities* for more information.

3.16. 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. The `gc_StartTrace() filename` parameter is ignored. The name of the log file is specified in the CCLIB_START_STRUCT data structure. See *Section 6.2. Populating and Using a CCLIB_START_STRUCT* for more information.

3. Applying Global Call Functions to Analog Loop Start Applications

3.17. gc_WaitCall()

The **gc_WaitCall()** function is not applicable when using the Dialogic® DM/V160-LP-HIZ Board.

4. Resource Allocation and Routing

Analog loop start protocols require a voice or tone resource for setting up a call. Application development considerations for using dedicated voice resources in an analog loop start environment are discussed in this chapter.

For Dialogic® Springware Boards with on-board analog loop start devices, a voice device and an analog loop start device comprise a single channel. Although these devices can be addressed separately, all analog signaling is processed by the associated voice device; analog signaling (ring detection and loop current detection) events are not transmitted over the SCbus. For Dialogic® DM3 Boards with on-board analog loop start devices, the voice device and the analog loop start device are separate devices.

Applications requiring voice resources during the entire call (for example, voice-mail, announcements) must have enough voice channels to dedicate one channel to each analog loop start channel. A single **gc_OpenEx()** function call can open both the analog loop start device and voice device on both Springware and DM3 Boards.

To perform activities such as routing and voice store and forward, use the **gc_GetVoiceH()** function to obtain the voice handle associated with a line device. For example, before playing a file, you can retrieve the voice handle using the **gc_GetVoiceH()** function. If needed, you may route other resources to the analog loop start channel (for example, to send a fax) and reroute the voice channel back to the analog loop start channel before setting up or waiting for another call. You must route the same voice channel back to the associated analog loop start channel on Dialogic® Springware Boards because these two resources were internally attached when opened. On Dialogic® DM3 Boards, no such restriction exists because the two resources are independent of each other.

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

Dialogic® Global Call Analog Technology Guide

```
/* Open a Global Call device with a voice channel and an
   analog loop start network time slot */
1  if (gc_OpenEx(&linedev, ":P_pdk_na_an_io:V_dxxxB1C1", 0, &usrattr)
    == EGC_NOERR) {
    /*
     * Wait for GCEV_UNBLOCKED event.
     */
    .
    .
    /* Make an outgoing call */
2   if (gc_MakeCall(linedev, &crn, "123456", NULL, 0, EV_ASYNC)
      == EGC_NOERR) {
      /*
       * Wait for GCEV_CONNECTED event.
       */
    } else {
      /* Process error from gc_MakeCall( ) */
    }
  } else {
    /* Process error from gc_OpenEx( ) */
  }
  .
  .
  .
```

Legend:

- 1 The **gc_OpenEx()** function:
- opens a Global Call line device using voice channel dxxxB1C1 and configures the line device to use North American Analog Protocol.
 - opens the analog loop start time slot and voice channel automatically.

SCbus time slot routing and attaching are done automatically. The function need only be called once for an analog loop start time slot/voice channel pair.

- 2 The **gc_MakeCall()** function is invoked once for each outbound call.

5. Analog Protocols

The protocols supported, protocol file naming conventions, protocol components, and their corresponding protocol files are described in this chapter.

NOTE: The information in this chapter is applicable to Dialogic® Springware Boards only. On Dialogic® DM3 Boards, the analog protocol is embedded in the firmware.

5.1. Protocols Supported

Protocols are distributed as part of the Dialogic® System Release Software. The Dialogic® Global Call protocols available are listed in the *Dialogic® Global Call Country Dependent Parameters (CDP) for PDK Protocols Configuration Guide*.

The protocol and parameters used at the application's interface to the PTT must complement those used by the local central office (CO). To maintain compatibility with the local PTT, Dialogic provides country dependent parameter files that can be modified to satisfy local requirements. User-selectable options allow customization of 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 are specified in the country dependent parameter (.cdp) file and may be modified at configuration time (that is, at any time before starting your application). The *Dialogic® Global Call Country Dependent Parameters (CDP) for PDK Protocols Configuration Guide* lists each supported protocol and describes the modifiable parameters in the protocol's .cdp file.

When using PDK protocols, some parameters are dynamically updateable (that is, the parameter value can be changed while the application is running). See *Section 2.14. Run Time Configuration of PDK Protocol Parameters* for more information.

5.2. Protocol File Naming Conventions

When a protocol is installed on your system, several files are installed, including the protocol module(s) and country dependent parameter files. The Global Call analog loop start protocol files use the naming conventions described in *Table 12*.

Table 12. Protocol File Naming Conventions

| Filename | Description |
|--|-------------------------------------|
| <i>pdk_cc_tt_d.psi</i> | PDK protocol state information file |
| <i>pdk_cc_tt_d.cdp</i> or <i>pdk_cc_tt_ffff_d.cdp</i> | country dependent parameter file |

where:

- **pdk** is a prefix for PDK protocols only.
- **cc** is a 2-character ISO country code, for example, na = North America.
NOTE: The country code na is used to designate protocols used in both the United States and Canada.
- **tt** is a 2-character protocol type, for example, an = analog protocol.
- **d** is a 1- or 2-character direction indicator. Valid directions are:
 - **i**: inbound
 - **o**: outbound
 - **io**: inbound/outbound
- **ffff** is optional and defines a special software or hardware feature supported by the protocol; 1 to 4 characters.

The protocol name used in the **devicename** parameter of the **gc_OpenEx()** function is the root name of the .cdp file (for example, *pdk_na_an_io* for North America).

Examples of the files included for the PDK North American analog protocol are listed in *Table 13*.

Table 13. PDK North American Analog Protocol File Set

| Description | Protocol Files |
|---|-------------------------|
| Analog loop start protocol module | <i>pdk_na_an_io.psi</i> |
| Inbound/outbound country dependent parameters | <i>pdk_na_an_io.cdp</i> |

5.3. Protocol Components

The file types included with a protocol are:

- protocol modules
- country dependent parameter (.cdp) files

5.3.1. Protocol Modules

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

For PDK, the protocol module is a protocol state information (.psi) file, a binary file that is interpreted by the PDK run-time component (PDKRT).

5.3.2. Country Dependent Parameter (.cdp) Files

In addition to the voice parameter file loaded by the `gc_LoadDxParm()` function, Global Call uses a country dependent parameter (.cdp) file that defines country specific and protocol specific parameters. For some protocols, certain parameters must be set in the country dependent parameter file to ensure proper operation of the protocol. Country dependent parameter files may be customized by modifying the file using any utility or word processor that can edit and save ASCII text. Refer to the *Dialogic® Global Call Country Dependent Parameters (CDP) for PDK Protocols Configuration Guide* for country dependent parameters that are most likely to be modified and for any required settings.

6. Debug Utilities

The Dialogic® Global Call debugging utilities are described in this chapter.

NOTE: The information in this chapter is applicable to Dialogic® Springware Boards only. There are no specific protocol debugging capabilities for Dialogic® DM3 Boards. Debugging of Dialogic® DM3 applications can usually be done with RTF logs.

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

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

The following sections discuss:

- Enabling and disabling the logging
- Populating and using a CCLIB_START_STRUCT
- Defining the GC_PDK_START_LOG environment variable
- Extended logging

6.1. Enabling and Disabling the Logging

In a Dialogic® Springware environment, the Global Call PDKRT (Protocol Development Kit Run-Time) provides a set of logging features that can be 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.

CAUTION

It is recommended that logging be done on an as-needed basis. Logging uses significant resources and can reduce the performance of the Global Call PDKRT call control library. Full logging (debug logging) enabled on many channels can reduce performance to such a degree that time-critical operations are affected and the behavior of a protocol may be altered.

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.

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

- Populating and using a **CCLIB_START_STRUCT**. See *Section 6.2. Populating and Using a CCLIB_START_STRUCT*.
- Defining the **GC_PDK_START_LOG** environment variable. See *Section 6.3. 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.

6.2. Populating and Using a CCLIB_START_STRUCT

The following code shows an example of how to define a **CCLIB_START_STRUCT**, populate the fields, and use it to enable logging when issuing the **gc_Start()** function.

```
GC_START_STRUCT t_GcStart;
CCLIB_START_STRUCT t_PdkStart;
t_PdkStart.cclib_name = "GC_PDKRT_LIB";
t_PdkStart.cclib_data = "filename: pdktest.log;
binaryfile: 1;
loglevel: ENABLE_DEBUG;
service: R2MF_ENABLE | CAS_ENABLE;
cachedump: WHEN_FULL | THREAD_ON;
channel: B1C1, B2C2-4;
cachesize: 10;
maxfilesize: 0;
```

6. Debug Utilities

```
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 value of the **cclib_name** field must be either **GC_PDKRT_LIB** or **PDGV_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 |
| service | See <i>Table 16</i> . | ALL_SERVICES or 0xFFFFFFFF |
| cachedump | See <i>Table 17</i> . | WHEN_FULL or 1 |
| cache size | Any positive integer | 1 (number of records in cache) |
| channel | See <i>Table 18</i> . | B*C* |
| maxfilesize | Integer | 0 (Megabytes) |
| mindiskfree | Integer | 20 (Megabytes) |

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

Dialogic® Global Call Analog Technology Guide

The following examples show how to set the **cclib_data** string:

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

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

Table 15 shows the valid values for the **loglevel** parameter.

Table 15. Loglevel Parameter Values

| loglevel | Valid Value | Description |
|---------------------------|--------------------|---|
| ENABLE_FATAL (default) | 5 | Only fatal errors are logged. A fatal error is an error that will make the program run abnormally or will stop the program. For example, in <i>channelimpl.cpp</i> , 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. |

6. Debug Utilities

| loglevel | Valid Value | Description |
|--|-------------|--|
| ENABLE_WARNING | 4 | All levels above ALERT are logged. An error occurs that may make the program run abnormally. For example, in <i>channelimpl.cpp</i> , the new local state is not ChanState_InService while the reason is Wait Call. An exception may be thrown, but log cache will not be dumped to a file automatically. |
| ENABLE_ALERT | 3 | All levels above INFO are logged. There is a problem, generally not an error, that the user should know about. |
| ENABLE_INFO | 2 | All levels above DEBUG are logged. Important information that the user needs to be aware of is logged. For example, in <i>channelimpl.cpp</i> , issuing a 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. |
| ENABLE_DEBUG | 1 | All levels are logged. This gives the most detailed information to help debug protocols or code step-by-step. For example, in <i>channelimpl.cpp</i> , a call to any of the GC_PDK_C_XXX functions should be logged at this level. Most routine logging should use this level. |
| Note: Values are in decimal but can also be specified in hex using a 0x prefix. | | |

Table 16 shows the valid values for the **service** parameter.

Table 16. Service Parameter Values

| service | Valid Value | Description |
|---|-----------------------|---------------------------------|
| ALL_SERVICES (default) | 0xFFFFFFFF (65535) | All services are enabled. |
| USRAPP_ENABLE | 0x00000001 (1) | Only USRAPP service enabled. |
| GCAPI_ENABLE | 0x00000002 (2) | Only GCAPI service enabled. |
| GCXLTR_ENABLE | 0x00000004 (4) | Only GCXLTR service enabled. |
| LINEADMIN_ENABLE | 0x00000008 (8) | Only LINEADMIN service enabled. |
| CHANNEL_ENABLE | 0x00000010 (16) | Only CHANNEL service enabled. |
| LOADER_ENABLE | 0x00000020 (32) | Only LOADER service enabled. |
| CALL_ENABLE | 0x00000040 (64) | Only CALL service enabled. |
| R2MF_ENABLE | 0x00000080 (128) | Only 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. |
| Note: Values prefixed with 0x are hexadecimal values. Decimal values are shown in parentheses. | | |

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

Table 17. Cachedump Parameter Values

| cachedump | Valid Value | Description |
|--|-----------------------|--|
| ON_FATAL | 0x0000 (bit 1 = 0) | The cache memory will be dumped to the log file once there is a log record with a FATAL level. |
| WHEN_FULL (default) | 0x0001 (bit 1 = 1) | The cache memory will be dumped to the log file once the log cache is full as determined by the 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 **channel** parameter.

Table 18. Sample Channel Parameter Values

| Example Value | Boards and Channels Enabled for Logging |
|----------------------|--|
| B*C* (default) | All boards and all channels |
| B-1C-1 | Only board number = -1 and channel number = -1 |
| B1C* | All channels on board 1 |
| B1C-1 | Only board 1 level |
| B1C1 | Channel 1 on board 1 |
| B1C1-5 | Channel 1 to 5 on board 1 |

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

6.3. Defining the GC_PDK_START_LOG Environment Variable

The GC_PDK_START_LOG environment variable can also be used to enable and configure logging.

The following examples show how to set the GC_PDK_START_LOG environment variable.

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

```
set GC_PDK_START_LOG = "filename : pdktest.log; binaryfile : 1;
loglevel: ENABLE_DEBUG; service : R2MF_ENABLE | CAS_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 GC_PDK_START_LOG environment variable definition:

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

This definition is equivalent to the logging configuration used in *Section 6.2. Populating and Using a CCLIB_START_STRUCT* and the definition for each field is also the same as described in that section.

6.4. Extended Logging

The `gc_ExtensionFunction()` function provides extended features directly from the call control libraries. For applications that use the PDK protocols, if logging is enabled, the `gc_ExtensionFunction()` function can be used to add user-specified log records to the log file.

6.4.1. `gc_ExtensionFunction()`

For debugging purposes, the `gc_ExtensionFunction()` should only be used if requested by Dialogic Technical Support. It enables users to include debug information useful to Technical Support personnel when reading the log file. The log file is a binary file that cannot be read without the required tools, which are supplied with the Protocol Development Kit Run-Time (PDKRT).

The function header of the `gc_ExtensionFunction()` function is:

```
gc_ExtensionFunction(int cclibid, LINEDEV linedev, CRN crn,  
                    void *datap)
```

where:

- **cclibid** is the Global Call call control library ID
- **linedev** is the Global Call line device handle
- **crn** is the call reference number
- **datap** is a pointer to a call control library-specific structure containing information about the extended feature

For extended logging, the **datap** parameter is a pointer to a structure of type `PDK_XTEN_LOG_FUNC`, which contains extended logging information. See *Section 6.4.2. PDK_XTEN_LOG_FUNC* for more information.

6.4.2. PDK_XTEN_LOG_FUNC

For extended logging, the **gc_ExtensionFunction()** uses the PDK_XTEN_LOG_FUNC data structure. The structure definition is as follows:

```
typedef struct
{
    PDK_XTEN_FUNCNUM    func_no;
    char*               log_data;
    PDK_LOG_LEVEL      log_level;
    PDK_SERVICE         service;
    char*               file_name;
    long                line_num;
} PDK_XTEN_LOG_FUNC;
```

Table 19 describes each field in the data structure.

Table 19. PDK_XTEN_LOG_FUNC Field Descriptions

| Field | Description |
|-----------|--|
| func_no | Identifies the extension feature requested. Possible values are: <ul style="list-style-type: none"> • PDK_FUNC_LOG = 1 • PDK_FUNC_DUMPLOG = 2 |
| log_data | A string that is to be added to the log file. |
| log_level | The logging level of the added record. Valid logging levels are: <ul style="list-style-type: none"> • PDK_LOGLEVEL_DEBUG = 1 • PDK_LOGLEVEL_INFO = 2 • PDK_LOGLEVEL_ALERT = 3 • PDK_LOGLEVEL_WARNING = 4 • PDK_LOGLEVEL_FATAL = 5 |

6. Debug Utilities

| Field | Description |
|-----------|--|
| service | The service name. Valid values are: <ul style="list-style-type: none">• PDK_SERVICE_USRAPP = 1• PDK_SERVICE_GCAPI = 2 |
| file_name | The name of the source file from which the log entry originated. |
| line_num | The line number in the source file from which the log entry originated. |

6.4.3. Extended Logging Code Example

The following code example shows how to include user-defined log records in the log file.

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

void main()
{
    GC_START_STRUCT gc_start;
    PDK_START_STRUCT pdk_start;
    PDK_XTEN_LOG_FUNC logstruct;
    char *data = "This is a log record";
    LINEDEV ldev;

    pdk_start.cclib_name = "PDKRT";
    pdk_start.start_parameters = "filename: pdkrt;
    loglevel: ENABLE_DEBUG";

    gc_start.nStartStructures = 1;
    gc_start.cclib_start_struct[0] =
        (CCLIB_START_STRUCTP)&pdk_start;

    gc_start(&gc_start);
    gc_Open(&ldev, ":NdtiB1T1:P_us_t1_em:" ,0);

    logstruct.func_no = PDK_FUNC_LOG;
    logstruct.log_data = data;
    logstruct.log_level = PDK_LOGLEVEL_ALERT;
    logstruct.service = PDK_SERVICE_USRAPP;
```

Dialogic® Global Call Analog Technology Guide

```
logstruct.file_name = __FILE__;  
logstruct.line_num = __LINE__;  
gc_ExtensionFunction(PDGV_LIB,ldev,0,&logstruct);  
/* PDGV_LIB is the ID of the PDKRT */  
/* the rest of the application goes here */  
}
```

Index

A

- additional tones, 20
- alarm handling, 38
- analog loop start, 16
- analog loop start device, 37, 61
- analog signaling, 18, 38, 61
- ANI information, 50
- answering machine
 - detect, 17
- answering machine detection, 50
- application
 - designing and coding, 15
- audio tones, 18

B

- block analog line, 18

C

- cadence break, 50
- call analysis, 17, 20
- call disconnect, 16
- call parameters, 17
- call progress, 20
- call progress tones, 17
- call termination
 - network, 38
- call transfer, 30, 32
- called party, 18

- caller ID, 50
- calling party, 18
- cause parameter, 49
- CCLIB_START_STRUCT
 - using for debugging, 68
- cdp file, 63
- channel-level parameters, 16
- code example
 - extended logging, 76
- connection types, 50
- country dependent parameter, 63

D

- debugging, 67
 - cclib_data fields and values, 69
 - enabling for PDK protocols, 68
 - log_cachedump values, 73
 - log_level values, 70
 - log_service values, 71
 - PDK protocols, 67
 - sample log_channel values, 73
- dedicated voice resources, 61
 - example, 61
- delete tones, 20
- destination CO, 18
- devicename, 54, 55
- devicename parameter, 64
- dialed digits, 18
- dialing code
 - case-sensitive, 51

Dialogic® Global Call Analog Technology Guide

- dialing mode, 51
- digit detection accuracy, 19
- directory number
 - DN, 49
- disconnect tones, 16
- disconnection
 - reason, 38
- DN
 - directory number, 49
- DTMF digits, 19
- DTMF signaling, 17, 19
- DX_CAP data structure, 16, 17
- dx_setevtmsk(), 45
- dx_setparm(), 16
- E**
- enhanced call analysis, 17
- enhanced call analysis parameters, 16
- event mask settings, 45
- extended logging, 75
 - code example, 76
- F**
- fax machine
 - detect, 17
- fax machine detection, 50
- frequency overlap, 19
- G**
- gc_AcceptCall(), 48
- gc_AnswerCall(), 16, 48
- gc_Attach(), 48
- gc_AttachResource(), 48
- gc_BlindTransfer(), 31
- gc_CallAck(), 17
- gc_CompleteTransfer(), 31, 34
- gc_Detach(), 49
- gc_DropCall(), 49, 57
- gc_ExtensionFunction(), 75
- gc_GetANI(), 49
- gc_GetCallInfo(), 22, 24, 49, 50
- gc_GetParm(), 50
- gc_GetVoiceH(), 61
- gc_LoadDxParm(), 16, 17, 65
- gc_MakeCall(), 17, 22, 34, 51, 62
- gc_Open(), 62
- gc_OpenEx(), 16, 54, 61, 64
- gc_ReleaseCall(), 56
- gc_ReleaseCallEx(), 56, 57
- gc_ResetLineDev(), 57
- gc_ResultValue(), 22, 24
- gc_SetParm(), 22, 57, 58
- gc_SetupTransfer(), 31, 34
- gc_Start(), 58
- gc_StartTrace(), 58
- gc_SwapHold(), 31, 34
- gc_WaitCall(), 16, 57, 59
- GCEV_ALERTING, 21
- GCEV_DISCONNECTED, 38
- GCEV_OFFERED, 16

GCEV_RESETLINEDEV, 57
GCPR_CALLPROGRESS, 22, 58
GCPR_MEDIADetect, 22, 24, 58
global tone detection, 20
GTD, 20

I

inbound call, 16, 18, 57
info_id parameter, 50
international networks, 19

L

LDID
 line device ID, 38
line device ID
 LDID, 38
local CO, 18, 63
log file, 67
logging
 extended, 75
loop current, 50
loop current change, 16
loop current detection
 analog signaling, 38, 61

M

MF digits, 19
MF signaling, 19
MF tone signaling, 17
multifrequency code, 19
multifrequency combinations, 19

N

naming convention
 protocol, 64
national networks, 19
network device independence, 20
network handle, 61
number of rings, 16
numberstr parameter
 dialing string, 51

O

options
 protocol, 63
outbound call, 17, 18

P

PDK protocols
 debugging, 67
 enabling debugging, 68
 programming guideline, 44
PDK_MAKECALL_BLK, 52
PDK_XTEN_LOG_FUNC, 75
PDKRT protocols
 call analysis, 25
Perfect Call, 17
pre-existing tones, 20
protocol, 63
 naming convention, 64
 service layer parameters, 41
 troubleshooting, 67
protocol module, 65
 ICAPI, 40
 PDK, 40
pulse dialing, 17, 18

Dialogic® Global Call Analog Technology Guide

R

- remote end, 48
- resource sharing, 38
- ring detection
 - analog signaling, 38, 61
- ringback tone, 18, 21
- rings parameter, 16, 48
- rotary dialing, 18

S

- SCbus, 62
- service layer parameters, 41
- setting up a call, 52
- signaling frequencies, 19
- signaling information, 18
- SIT, 20
 - special information tones, 20
- special information tones
 - SIT, 20
- supervised call transfer, 30, 32

T

- telephone number
 - called party, 18
- timeout, 51
- tone definition, 20
- tone ID, 20
- tone resource, 61
- tones downloaded, 20
- troubleshooting, 67

U

- unblock analog line, 18

V

- voice channel, 20
- voice channel parameter (.vcp)
 - ASCII text file, 16
- voice detection, 50
- voice device, 61
- voice handle, 61
- voice resource, 61
 - dedicated, 61