GDK Version 5.0 Programming Reference Manual for Windows

05-6025-002

COPYRIGHT NOTICE

Publication Date: August, 2000

Part Number: 05-6025-002

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

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

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

OPERATING SYSTEM SUPPORT

The term *Windows* refers to both the Windows NT® and Windows® 2000 operating systems. For a complete list of supported Windows operating systems, refer to the *Release Guide* that came with your Dialogic System Release for Windows, or to the Dialogic support site at *http://support.dialogic.com/releases*.

Table of Contents

List of Tables

List of Figures

Preface

Purpose

The purpose of this manual is to describe Dialogic's GDK system architecture, detail the programming models and their associated Application Programming Interface (API) function calls, and provide instruction on fax application programming.

Audience

This manual is designed for fax and voice software programmers developing on the Microsoft Windows platform. Familiarity with computer telephony, C-language programming and Microsoft Windows software development is highly recommended.

Using This Guide

If you are new to fax application development, read Chapter 1 for an overview of fax technology and Chapter 2 for an introduction to the fax architecture.

If you are a fax developer new to Dialogic CP Fax technology, read Chapter 2 before reading Chapters 4 and 5.

If you are already developing fax software using the DOS, OS/2 or UNIX versions of GDK, read Chapter 5 for programming model information and function call descriptions.

Conventions

The conventions used in this manual follow:

- Code fragments are shown in courier text:

  ```
  gfqFindFirst( )
  ```

- Function calls are shown in Arial text:

  gfqFindFirst( )

Notes and cautions are shown as follows:

NOTE: Text of note

CAUTION: Text of caution

Reference Documentation

- *Error and Status Codes Manual*
- *GDK Installation and Configuration Guide for Windows*

1. Introduction to Fax Technology

Overview

This chapter provides a brief overview of fax technology. This chapter discusses the following:

- History of fax
- Understanding fax technology
- Advantages of computer-based faxing
- Standards
- Fax applications

History

Although the first successful fax was actually patented in 1843, it wasn't until the 1930s that fax systems had evolved into the form we recognize today. In 1966 the first fax standard was adopted: *EIA Standard RS-328, Message Facsimile Equipment for Operation on Switched Voice Facilities Using Data Communication Equipment*. This standard made more generalized business use of the fax possible. It became known as the Group 1 standard.

In 1978, the Consultative Committee for International Telephone and Telegraph (CCITT) came out with the Group 2 recommendation. The fax had achieved worldwide compatibility, and this led to a more generalized use of fax machines by businesses and the government.

By the time the Group 3 standard arrived in 1980, fax was well on its way to becoming the everyday tool it is today. This digital fax standard opened the door to reliable high-speed transmission over telephone lines.

Understanding Fax Technology

The major elements of fax technology include:

- Elements of a fax call
- Compression
- Resolution
- Routing

Elements of a Fax Call

The fax call elements consist of five phases:

- Establishing the call
- Pre-message procedure
- In-message procedure and message transmission
- Post-message procedure
- Releasing the call

Establishing the Call (Phase A)

The first stage occurs when the transmitting and receiving units connect over the telephone line, recognizing each other as fax machines.

Pre-Message Procedure (Phase B)

In this stage, the answering machine identifies itself in a burst of digital information packed in frames conforming to the High-Level Data-Link Control (HDLC) standard. The caller then responds with information about itself.

In-Message Procedure and Message Transmission (Phase C)

This is the actual fax transmission part of the procedure. The in-message procedure and message transmission occur simultaneously. The in-message procedure deals with synchronization, line monitoring, and problem detection. Message transmission is the actual data transmission. Once a page/file has been transmitted, the next phase begins.

Post-Message Procedure (Phase D)

After a page has been transmitted, the sender and receiver revert to the pre-message procedure modulation rate. If the sender has more pages to transmit, the in-message procedure and message transmission (Phase C) begins again for the next page. After the last page is sent, the sender transmits either an End of Message (EOM) frame, or an End Of Procedure (EOP) frame to show it is ready to end the call. The receiver then sends a confirmation.

Releasing the Call (Phase E)

Once the call is complete, the side that transmitted the last message sends a Disconnect (DCN) frame and hangs up without waiting for a response.

Compression

One of the most important components of a successful modern fax call is compression technology. The various compression encoding schemes used for fax remove redundancy from scanned material and restore the data at the receiving end. Using any of the compression schemes shortens transmission times and reduces errors.

Resolution

Fax images are made up of dots. Resolution refers to the size and density of the dots used to portray an image. There are two resolutions widely used by fax machines and fax boards: standard and fine. Fine mode contains twice as many dots per inch than the standard mode, which means the quality of the image on the receiving end is clearer. However, with more dots per inch, the file is bigger, and it takes longer to transmit the file across the telephone line.

Routing

Recent enhancements such as fax security and private mailboxes have created a need for a routing mechanism. Three types of routing mechanisms include:

- Dual-Tone Multi-Frequency (DTMF)
- Direct Inward Dialing (DID)
- T.30 subaddress

DTMF Routing

DTMF routing uses the buttons on a touch-tone telephone. The disadvantage of DTMF is that the sender needs an extension number as well as a telephone number.

Direct Inward Dialing

DID is generally considered the most foolproof, transparent routing alternative. All the sender does is dial a single telephone number, and the fax is sent directly to the recipient's workstation.

T.30 Subaddressing

A subaddress encodes a numeric string identifying the recipient into the information exchange that occurs in Phase B of the fax call.

Advantages of Computer-Based Faxing

Computer-based faxing (CBF) allows PC users to send and receive faxes using graphic and text files. Advantages of computer-based fax include:

- High-quality output
- Convenience
- Progress monitoring
- Time and effort savings

High-Quality Output

CBF generally provides a higher quality document than the traditional fax machine. With CBF, the computer can convert the document into an image without degrading the sharpness, which can occur with the scanners in fax machines. This conversion improves the faxed image's appearance.

Convenience

CBF is more convenient to use than fax machines, especially for documents created or stored in computer systems. It is easier to send a fax directly from the

computer than to print a copy of the document and manually send it from a fax machine.

Progress Monitoring

CBF can monitor the progress of the outgoing fax transmission, and give you status on each transmission depending on the outcome of the call (i.e., sent, busy, failed, etc.).

Saves Time and Effort

Computer-based fax saves valuable time and effort. The name and fax numbers of recipients only need to be entered once in the computer. Then, any time you want to send a document to someone whose “fax address” is stored in the computer, you can easily select the name in the computer instead of re-entering the phone number into the fax machine. This feature is especially useful when sending out multiple copies of documents; no more standing at the fax machine sending faxes one at a time.

Standards

The ITU-T is one of four permanent parts of the International Telecommunications Union (ITU), based in Switzerland. It issues recommendations for standards applicable to modems and other areas. The standards it recommends are generally accepted and adopted by the fax/modem industry.

In order for a fax device to be allowed to connect to the public telephone system in another country, it must first be approved by the national Post Telephone and Telegraph (PTT) administration. Standards are available from this website:

http://www.itu.ch/.

Fax Applications

As fax technology continues to develop, new applications appear to fill the needs of users. The major growth areas for CBF include:

- E-mail Fax Gateways
- Mini/Mainframe Fax Servers
- Fax Store-and-Forward Systems
- Image Systems
- Integrated Voice/Fax Systems
- Vertical Fax Applications
- Public Fax Services

2. GDK System Architecture

About GDK for Windows

GDK consists of a fax channel and communication software, which provides many features and capabilities for fast, convenient fax transactions. Some of the standard features include:

- Sending one or more faxes to multiple locations (broadcasting)
- Sending files using binary file transfer (T.434)
- Recording the status of incoming and outgoing faxes
- Sending and receiving a fax on the same phone call (turnaround polling)
- Setting the fax transmission rate
- Delaying fax transmission
- Using T.30 subaddressing

The GDK system software provides the subsystems required to develop a fax application. The system software includes a fax subsystem compatible with Windows NT. The fax subsystem consists of the Dispatcher, the Queue file, the firmware, and the programming tools to control these components. The GDK contains the functions you need to build a fax application.

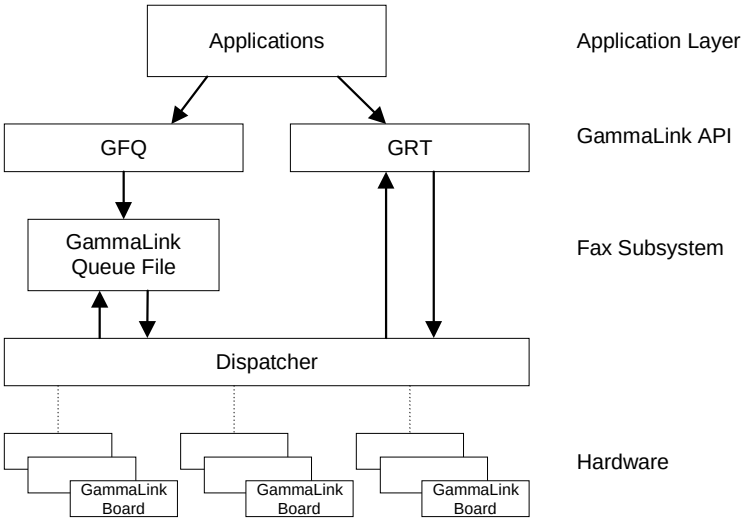

Figure 1. GDK Architecture

Refer to Chapter 5, “Programming Models”, for a complete description of the base GDK functions.

Refer to Chapter 6, “Developing with PEB”, for a description of the functions needed for PEB routing.

Refer to Chapter 7, “Developing with SCbus”, for a description of the functions needed for SCbus routing.

Refer to Chapter 8, “Fax Status Files”, for a description of GDK system status and configuration functions.

Minimum System Requirements

The following hardware and software are required:

- Dialogic CP Fax Series board (s)
- GDK for Windows software, version 5.0 or later

- Pentium® system
- Microsoft Windows NT operating system, (workstation or server) version 4.0 or later, or Microsoft Windows 2000

About the Dialogic CP Fax Series Hardware

Each fax channel (or cell) consists of a microprocessor-based facsimile modem, RAM, CPU, and a telephone network interface. These components allow each channel to function as an independent, computer-based subsystem interfacing with the host computer.

There are two types of CP hardware: analog and digital. The analog CP boards have lower channel density and contain on-board telephone network connectors. The digital CP boards have much higher channel density, but require a separate telephone network interface device to communicate with the PTT system.

Most of the analog CP product line is available internationally, depending on the homologation status of the hardware model. The digital boards do not need homologation certification, as they do not have an on-board telephone connector.

Firmware Features

The faxcell firmware included with the software is for use in the US and in international countries. Refer to the Documentation directory of the product CD for additional updated information about the firmware.

GDK System Components

Device Drivers for ISA and PCI

The software contains two kernel-mode device drivers. The ISA bus device driver, *glfxisa.sys*, only communicates with the Dialogic CP Fax Series ISA boards; and the PCI bus device driver, *glfxpci.sys*, only communicates with the Dialogic CP Fax Series PCI boards.

Firmware Download Utility

The firmware download utility, *%gfax%\glfxdldr.exe*, is used by the Dispatcher during service startup to download the GDK fax channels. This utility can also be run from the command-line in a console window. For trouble-shooting purposes, the option '-v' can be appended to the utility name to see verbose output when running from a console window. The filename stored for the firmware command can be configured with the Dialogic GDK Configurator utility. Refer to the *GDK Installation and Configuration Guide for Windows* for more information.

The GDK System Service (the Dispatcher)

The GDK System Service (also called the Dispatcher Service) is the operating system dependent interface to both the fax channels and to the API library. This service can be configured from the Control Panel Services Applet (Figure 2).

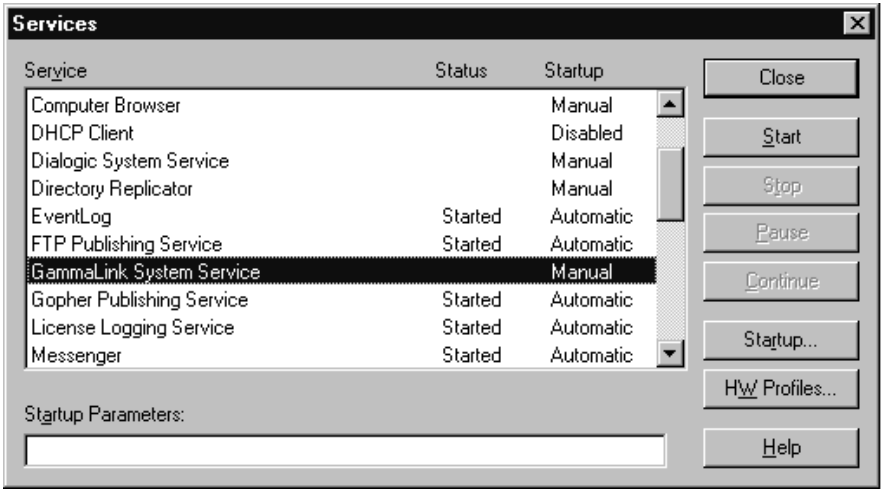

Figure 2. System Services Control Panel

You can start and stop the GDK System Service from the Control Panel Services Applet, the Start GDK System Service Program in the GDK program group, or from a Command Prompt console window.

The GDK System Service outputs information to the following files:

%GFAX%\gfdNT.log	Provides startup and shutdown status information, as well as automatic board detect results and configuration files created.
%GFAX%\gfax.$co	Dispatcher "stdout" output. This file is kept open until the Dispatcher exits.

GDK Dispatcher as a Windows System Service

There are two processing models: batch processing mode and interactive processing mode. The Dispatcher acts as a fax scheduler in the batch processing mode, finding an available line on any GDK port in the chassis and sending records to the CP Fax board for transmission.

In the interactive processing model, the Dispatcher is the intermediary between the application and the GDK ports. The application is the fax scheduler. It can be programmed to NOT post the completed queue record to the queue file.

The Dispatcher supplies queue records to the firmware. It also can reserve buffer space in RAM for these records to improve system performance. The number of record buffers in a system is configured by the BUFFERS command. The recommended buffer number is two per fax channel, and each record requires 516 bytes.

The QUEUET command determines the time in seconds that the Dispatcher scans the Pending List for new jobs. When the QUEUET timer expires, the Dispatcher starts at the beginning of the Pending List. It then loads and marks “BUSY” as many queue records that are ready for transmission as it has buffers available.

When the record is passed onto the channel, an additional bit is set indicating that the record is off host. When the system sees a record with this bit setting, it does not disturb the record while it is in a buffer or being processed by the firmware. Several records can be marked BUSY even when there is only one fax channel in the system.

Although the original record may be undisturbed in the Queue File, a record marked “BUSY” indicates that a copy of the original is in the record buffer of the

Dispatcher or is held by the firmware. The Dispatcher overwrites the original record with updated information when the transaction completes.

Once a remote site confirms a transmission, the Dispatcher sends a confirmation to the Queue file. If a transmission fails, the dispatcher notifies the Queue file, which records the appropriate error message for the failed transaction.

Network Drive Access

The GDK System Service must be configured to provide access to network drives using drive letter notation (i.e., "p:\public\fax\received\..."). These network drives can be on either Novell or Microsoft servers. For the network drive access to work, the Dispatcher has to be logged into the server where files are located (for sending), and where they will be located (for receiving). This task must be performed *after* the installation completes. Follow these steps to log the Dispatcher into the server:

1. Activate the Control Panel icon and select Services.
2. Select GDK System Service. After pushing the Startup button, a dialog box appears, allowing the user to select the Startup Type (Automatic, Manual or Disabled). Below these choices is the section called Log On As. The default install specifies “System Account.”
3. Choose the radio button for “This account.” This activates the three entry boxes. Type in a user name whose account allows access to the server in the first box. The second box requires the password, and the third is password confirmation box.

The GDK System Service

The GDK Service depends on the successful start of the device driver service and, in certain configurations, the Dialogic service. The GDK System Service configures the fax channel and the GDK system using the information stored in the registry.

GDK Service Dependencies

The GDK will configure the GDK System Service dependencies based on the CP Fax Series hardware detected in the system. The GDK System Service depends on:

- ISA driver service, *glfxisa*, if ISA fax hardware is detected
- PCI driver service, *glfxpci*, if PCI fax hardware is detected
- Dialogic service, if digital fax boards (i.e. fax hardware with CP4/SC, CP6/SC or CP12/SC board types) are detected

The hardware driver service dependencies are required for communication with the CP Fax Series hardware. The Dialogic service dependency is required for an SCbus-system configuration.

About the Firmware

The firmware is the on-board software that manages the fax transactions and image conversions for each fax channel. The firmware drives the GDK system, telling the GDK System Service when it is ready for more work and when it needs to be serviced. The firmware is downloaded on to each fax channel during GDK system startup.

About the Queue File

The Queue File operates as the database for the Fax subsystem in the batch processing mode. It stores the fax transmission(s) until the Dispatcher requests it. The Queue File stores future transactions as well as results of past transactions. Transmission statistics regarding each completed transaction are stored in the queue record. For more information about the Queue file, see Chapter 4.

GDK System Features

TIFF Capability

GDK supports Group 3 1-D and 2-D T.4, and Group 4 T.6 compression. The type of compression may be selected for both sending and receiving fax files on each fax channel individually. The selection is controlled by the following: GFXFORM, GFXSTWOD, GFXRT6, GFXRTWOD, GFXST6.

When sending files, GDK supports these formats:

- ASCII text
- TIFF Type 3
- TIFF Type 3 2-D
- TIFF Type 4
- PCX

The file format is determined by the header of the data.

Tag Fields Supported

GDK currently supports and produces the tags for the TIFF fields listed in Table 1.

NOTE: Only one TIFF strip is generated per fax page. Each page may be a separate, sequentially-named file or a multiple-image TIFF file.

Table 1. GDK TIFF Tags

Tag No.	Name	Data Type	Description
---[1]	ByteOrder	int[2]	Order of bytes: II (0x4949) = Intel; required for GDK MM (0x4D4D) = Motorola
254	NewSubFileType	long[3]	A 32-bit flag indicating the type of data contained in this subfile.
256	ImageWidth	long[2]	Number of pixels per scanline. 1728 for A4 or letter 2432 for A3 2048 for B4
257	ImageLength	long[2]	Number of scanlines in image.
258	BitsPerSample	unsigned[2]	Number of bits per pixel sample; only “1” for fax.

Tag No.	Name	Data Type	Description
259	Compression	unsigned[2]	Type of compression. Possible values: 1 = no compression (not supported) 2 = CCITT G3 1-D, no EOL (not supported) 3 = CCITT 1-D, with EOL or 1-D/2-D combined 4 = CCITT G4 5 = LZW compression (not supported) 32773 = PackBits (not supported)
262	PhotoInterpret	unsigned[3]	Photometric interpretation. Possible values: 0 = black on white (Default) 1 = white on black 2 = RGB scheme (not supported)
266	FillOrder	unsigned[2]	Order of the image bits, in bytes. Possible values: 1 = most significant bits are filled first 2 = least significant bits are filled first (Default)
269	DocName	char[3]	The name of the document from which this image was scanned.
270	Description	char[3]	A comment about the image.
273	StripOffset	long[2]	For each strip, the byte offset of that strip with respect to the beginning of the TIFF file.
277	SamplesPerPixel	int[2]	Number of samples need to define a pixel. Possible values: 1 = monochromatic data (Default) 3 = color data (not supported)

Tag No.	Name	Data Type	Description
278	RowsPerStrip	long[2]	Number of scanlines per strip. Multiple strips are not supported.
279	StripByteCounts	long[2]	Length of a strip.
282	X_Resolution	int[2]	Pixels per resolution unit in the horizontal direction. Default = 204.
283	Y_Resolution	int[2]	Pixels per resolution unit in the vertical direction. Default values: 98 = standard resolution 196 = fine resolution
286	OffsetX	long[3]	Offset of the left side of the image, with respect to the left side of the page, in Resolution Units.
287	OffsetY	long[3]	Offset of the top of the image, with respect to the top of the page, in Resolution Units.
292	Group3Options	long[4]	32-bit flag describing options. Possible bit values: 0 = 1-D compression used (Default) 1 = 2-D compression used 2 = uncompressed data may be used (not supported) 4 = guaranteed byte alignment
293	Group4Options	long[5]	32-bit flag describing options. Value must be zero.
296	ResolutionUnit	int[3]	Unit of measurement to be used with X_Resolution and Y_Resolution. Possible values: 1 = no unit of measurement (not supported) 2 = inches (Default) 3 = centimeters

Tag No.	Name	Data Type	Description
297	CurrPageNum	int[3]	Page number and number of pages in a multiple-page file.
326	BadFaxLines	long[3]	Number of scanlines with an incorrect number of pixels (TIFF Class F).
327	CleanFaxData	int[3]	Describes how the data was cleaned. Possible values (TIFF Class F): 0 = data contains no lines with incorrect pixel counts or regenerated lines (Default) 1 = lines with incorrect pixel count were regenerated on receipt 2 = lines with incorrect pixel count existed, but were not regenerated by receiving device
328	ConsecutiveBad FaxLines	long[3]	Maximum number of consecutive fax lines that contain an incorrect number of pixels (TIFF Class F).

1 Not defined as a tag

2 Required for GDK

3 Optional

4 Recommended for compression 3 only (see TIFF Tag No. 259)

5 Recommended for compression 4 only (see TIFF Tag No. 259)

Filenaming Convention

GDK has a number of filenaming conventions, which depend on the type of file. The following discusses these conventions.

Received Faxes

A received fax file is given the following default name:

a001p001.tif, where “a001” is the fax-call number, “p001” is the page number of the document, and the extension “tif” indicates that this is a TIFF file. For example:

a004p008.tif	indicates the fourth call received and the eighth fax page
a002p001.tif	indicates the second call received and the first fax page
a004p004.tif	indicates the fourth call received and the fourth fax page

In multiple-channel systems, the first letter will be an “a” for channel 1, “b” for channel 2, and so on. Depending on the filename format, multiple pages can be received per call (Table 2). The default filename format can be set by the GFXRECVPATH command or through the queue record filename and operation fields. The operation field must be set to ANSWER_DEFAULT. For more information, see Chapter 4.

Table 2. Filename Formats for Receiving Multiple Pages

Filename Format	No. of Page/Call
f0001p01.tif	Up to 99
f001p001.tif	Up to 999
f01p0001.tif	Up to 9999

Receive Faxes in Multiple-Image TIFF Files

Multiple-page faxes may also be saved in one multi-page TIFF file. This option is turned on using the gfccontrol 36 command. For more information, refer to Chapter 3.

Multiple-Page Documents

The GDK channels automatically sends any filename and includes sequentially-numbered pages without them being explicitly specified. For example, to fax a three-page document named:

file001.tif
file002.tif
file003.tif

only the filename of the first page (file001.tif) should be specified for the filename to send.

If the filenames of all three pages are specified, the recipient will first receive these pages:

file001.tif
file002.tif
file003.tif

then:

file002.tif
file003.tif

and finally:

file003.tif

This is because GDK automatically looks for subsequently named files.

Controlling Next File Send Option

This feature lets you indicate whether to send the next file in the file sequence. When files are stored, each page associated with the file is saved using a numbering sequence (i.e., fax001.TIF, fax002.TIF, or data001.DAT, data002.DAT etc.).

You can specify whether or not to send the next file in the sequence by appending a command-line option to the end of the filename. This feature is enabled or disabled on a job-by-job basis.

Table 3 shows the next file send command options.

Table 3. Next File Send Options

File Type	Command Line Options	Description	Default Setting
Single-page Fax File Sequence	-NP0	Disables sending of the next file in the sequence	
Single-page Fax File Sequence	-NP1	Enables sending of the next single-page file in the sequence. Specifies that other files in the sequence will be sent (if any others exist).	Default single-page fax file setting
Multi-page Fax File Sequence	-NP0	Disables sending of the next file in the sequence	Default multi-page fax file setting
Multi-page Fax File Sequence	-NP1	Enables sending of the next file in the sequence. Specifies that other files in the sequence will be sent (if any others exist	
BFT File Sequence	-NP0	Disables sending of the next file in the sequence	Default BFT file sequence setting

File Type	Command Line Options	Description	Default Setting
BFT File Sequence	-NP1	Enables sending of the next file in the sequence. Specifies that all other file in the sequence will be sent (if any others exist).	

Binary File Transfer

GDK supports Binary File Transfer, which conforms to the T.434 BFT standard protocol. BFT reception can be enabled for T.434 BFT on a per-channel basis with the following command:

```
GFXFAXCONTROL 1020 1
```

or on a per-job basis by setting the queue record protocol field.

BFT transmission can be enabled per channel with the following command:

```
GFXFAXCONTROL 1021 2
```

or on a per-job basis by setting the queue record protocol field.

NOTE: The GFXECM command must be set in order for BFT to work. If file transfer is used, the received filename will be a list of files in the format xxxxXFER.FLS. This list will contain the name of the file that was received, which will be the same as the original sent filename, as long as the DOS xxxxxxx.yyy format is observed, and the filename does not already exist on the target drive. The “f001p001” filenaming style is used if the sent filename already exists, but the original filename will be included in parentheses next to the filename that was written in the list. For example, A001XFER.FLS could contain:

c:\RECV\TEST.TXT
c:\RECV\A001P001.TIF (TEST.PCX)

Routing

Recent enhancements such as fax security and private mailboxes have created a need for a routing mechanism. Four types of routing mechanisms include: Dual-Tone Multi-Frequency (DTMF), Direct Inward Dialing (DID), T-1 digit collection, and a T.30 subaddress.

About the DTMF Capability

Most CP Fax Series boards have the ability to decode and store incoming touch-tone (DTMF) digits. With the ability to capture incoming DTMF digits, applications can be built to utilize this information for specialized tasks, such as sending out information in response to incoming calls.

For example, if an information service wanted to fax weather maps on request, it could assign DTMF digits to specific geographic areas. A client wanting a weather map of Northern California, for example, would be instructed to call a certain fax number and, at the tone, enter the assigned DTMF digits. GDK would receive the call and record the digits. The application would check the GDK records, load the transaction, and send the map to the caller's fax machine. Additionally, applications such as a network fax server can use this feature to provide security of information, because a workstation may be assigned specific digits. For example, only the intended recipient can view faxes sent to these digits; thus, access to incoming information can be controlled.

How GDK Works With DTMF

When a call is made to a CP Fax Series board that can detect DTMF tones, the board responds with a tone that is different from the fax tone, which signals the caller to enter the DTMF digits. After the DTMF digits are entered, the board responds with the fax tone, which signals the caller to activate the fax machine to send the fax. The CP Fax board receives the fax, and terminates the connection. If no tones are received, it waits for a designated time period and then continues with fax tone, assuming that a fax is on the other end.

After the connection is terminated, the fax board creates a queue record containing details of the transaction, including the DTMF digits. If the Dispatcher is running in batch processing mode, the queue record is then filed in a database called the Queue File. If the Dispatcher is running in interactive processing mode, the queue record is sent directly to the application. From there, an application takes over and uses the information stored in the queue record to carry out the next task.

To take advantage of the DTMF digits, callers must be instructed to use the appropriate digits at the DTMF tone. However, if a caller or an unattended fax machine transmits at the DTMF tone, GDK will accept the transmission and post the transaction without digits to a queue record. If this occurs, the fax must be routed manually.

Using the DTMF Capability

DTMF digit collection is a channel-specific feature. The following commands need to be set for each channel to enable the use of DTMF:

GFXDIGITS	Sets the number of DTMF tones that can be entered by a sender
GFXDTMFTIMEOUT	Sets the timeouts for waiting for DTMF input
GFXDTMFTONE	Specifies DTMF tone to issue when answering a call

Storing Routing Digits

When a fax is received, GDK stores the DTMF digits in the user_id field of the corresponding queue record in the Queue File. Then, an application can use this information for specialized tasks. A network fax server, for example, could use these digits to route incoming faxes.

The user_id field can contain a total of 34 characters. However, storage for only 24 characters is available, because other information, such as the user identification, may be stored in this field. (If no user identification is specified, “SYSOP” is used as the default.) GDK can handle up to 63 incoming digits; however, this will not be necessary in most cases. When it is necessary to store a large number of routing digits (such as a credit card number or a security code), this information will be stored on disk as a file. If the number of digits received is larger than the queue record can hold, a file will be created containing the digits followed by a null character. The decision to create a file is made on a call-by-call basis, because terminated input may not contain the quantity of digits specified by the GFXDIGITS command.

The name of the file that is created is appended to the user_id field as follows:

;D=@<filename>

where "@" signifies that a filename is being given. The filename will be in the form:

F000001.DGT

where "F" is the channel number. The filename digits increment for each subsequent transaction.

Direct Inward Dialing (DID)

Direct Inward Dial (DID) capability is only available to CP Fax Series hardware with a DID interface. Refer to the CPD/220 hardware installation manual for further information on configuring your system for use with DID lines and about DID lines in general. The major advantage of routing using DID service is simplicity for the caller. The caller dials a single number and the central office (CO) extracts the routing digits from the inputted number, rather than have the caller input the digits at the DTMF tone after dialing the fax number. Enabling DTMF capability and the process of storing the routing digits are the same as described above, with the exception of no tone being needed to prompt the user. The GFXDTMFTONE command should not be used with DID setups.

T-1 Digit Collection

The CP Fax Series board is also capable of collecting digits automatically from the telephone company using a T-1 trunk. A CP4/SC, CP6/SC, or CP12/SC board is connected to a T-1 interface board via a PEB in a fax-only system (i.e., no voice boards on the PEB). The most common telephone interface board for this configuration is the Dianatel EA24. The commands to set up digit collection are the same as for the DID, including not using the GFXDTMFTONE command.

T.30 Subaddressing

T.30 subaddressing allows a string of characters and numbers to be sent with the fax. This string is known as the subaddress. It also allows fax servers to do routing based on this subaddress.

GDK supports T.30 subaddressing, but it is disabled by default. Add the following command to enable T.30 subaddressing for each channel requiring this routing capability:

```
GFXFAXCONTROL 71 1
```

Sending a Subaddress

The subaddress is appended to the end of the dialing string starting with a #.

NOTE: The # is the default string delimiter. The start string delimiter can be configured using the GFXFAXCONTROL 73 command.

For example, to send a fax to 555-1212 with a subaddress of 9873, the appropriate dialing string should be: 555-1212#9873.

Receiving a Subaddress

The subaddress is stored in the USERID field of the queue record. To indicate that the USERID field contains a subaddress, the USERID field will start with S:=.

For example, if a fax was received with a subaddress of 9876, it would be represented in the USERID field as SYSOP;S:=9876.

Recording Line Noise

In the Answer-and-Receive mode, GDK can record the line noise and the status of telephone-line signals received during training. This information can then be used to determine the quality of a telephone line. If there is any fluctuation in the readings, be sure to receive a sufficient number of faxes to determine whether a problem is transitory or chronic.

Transparent PRI Support

Transparent PRI Support is an easy way to support the High-Density PRI solutions. This allows FSP developers to create ISDN PRI solutions to handle the PRI interfaces and the FAX resources without re-writing application code. The registry parameters that have been added to provide the Transparent PRI Support are defined in the New Parameters section in Chapter 3, Configuration Commands of this guide.

GDK System Configuration

The configuration parameters for the GDK software are now stored in the registry. The configuration can be changed using the Dialogic GDK Configuration utility.

Control Panel Configuration Utility

The automatic board detection and configuration processes are controlled by the services control panel applet. For more information, refer to the *GDK Installation and Configuration Guide for Windows*.

Setting the Country Code Properly

The COUNTRY command is required by the GDK firmware for proper operation of the fax channel. The COUNTRY parameter defines the country where the fax board has been designed to operate in, *not the country where the board is currently installed*. The firmware needs this information to select the correct electrical characteristics and PTT modem parameters.

NOTE: The correct COUNTRY parameter code should be the country or countries specified on the hardware packaging materials. If the packaging materials specify more than one country, set the COUNTRY parameter code to the country in which the board is installed (the country must be one of the approved countries).

If the fax board is connected to a programmable PBX, the correct COUNTRY parameter code will depend on the following:

- the line characteristics generated by the PBX
- the country or countries for which the board has been designed

If the fax board has not been designed to recognize the line characteristics of the telephone network connection, it will not function properly.

Country Codes and Call-Progress Error Codes

Country codes are the prefixes used for international calls. They also serve as configuration values for the GDK software, setting up the date strings that are in the correct format for the specified country and activating country-specific dialing schemes in the software. GDK provides call-progress error codes that can help

debug international calls. Refer to the *Error and Status Codes Manual* for a list of error codes.

Enabling Debug

Traditionally the GDK System Service has only allowed debug logging to a text file called %GFAX%\GFAX.$DL. Enabling debug is accomplished by the following command:

```
HKEY_LOCAL_MACHINE\SOFTWARE\GammaLink\_global

    DEBUG
```

The DEBUG command is configured by one or two parameters whose mask values are defined in Table 4 and Table 5.

In addition to the DEBUG command, more options are available. With this release, the GDK System Service logs debug trace information to shared memory and only saves debug log information to file, if configured to do so. The level of trace information logged to shared memory and the level of trace information logged to a text file is separately configurable through various bit mask values. In Transparent PRI mode, the GDK system also provides access to Dialogic's PRI Trace logging functionality.

In addition to the existing DEBUG command, logging is now enabled and configured by setting the following commands:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Dialogic\Gammalink\Debug

    UseSRAM

    SRAMMask

    LogFile

    LogFileMask

    DebugToSRAM

    UseGFAX$DL
```

Debugging is enabled by first setting the UseSRAM command to a string value of either "yes" or "no".

NOTE: If UseSRAM is set to "no", none of the other debug commands, including the existing DEBUG command, are enabled.

The level of debug information sent to shared memory is configured by setting SRAMMask to a bit mask whose value is defined by OR'ing the bits specified in Table 6. Saving debug trace information to a text file is enabled by setting the LogFile command to a full path and filename. The level of debug information written to the text file is configured by setting the LogFileMask to a bit mask whose value is defined by OR'ing the bits specified in Table 6. See GDK 5.0 New Parameter Summary for more information about these commands.

NOTE: Do not save log information to a text file during normal operations; as it is disk intensive and will grow too large.

The functionality of the DebugToSRAM and UseGFAX$DL commands depend on the traditional DEBUG command being enabled. Setting the DebugToSRAM command to "yes" will include traditional DEBUG information in shared memory. Setting UseGFAX$DL to "yes" will save only traditional DEBUG information to a text file called %GFAX%\GFAX.$DL. These two commands are provided to maintain backward compatibility with the traditional DEBUG mechanism of logging to a text file.

The traditional DEBUG command provides system level and fax call progress diagnostic information. The level of detail and category of information provided is determined by the DEBUG command's mask value. The DEBUG command can contain two parameters; the second is optional. The first parameter specifies system level trace information; the second parameter specifies channel (or firmware) level trace information.

The debug mask value specifies the type and level information written to the log file. Table 4 defines the mask values for the DEBUG command's first parameter. Table 5 defines the mask values for the DEBUG command's optional second parameter.

NOTE: The DEBUG command requires the high-order byte and the low-order byte of each parameter to be the same.

Table 4. Mask Values for DEBUG Parameter 1

Category	Mask	Description
Global diagnostics	0x01	Enables the logging of serious error messages.
Board messages	0x02	Enables board debug messages.
Polling diagnostics	0x04 = Type only 0x08 = Type and parameters 0x0C = Full trace	Reports every request the board makes of the System Service. Reports additional parameters for each transaction. Reports every poll along with details of internal packet.
Messaging diagnostics	0x10 = Connections 0x20 = Transactions 0x30 = Full trace	Reports the results of event notification and remote status and control operations. (It is recommended that 0x30 not be used.)
Queue file diagnostics	0x40 = Transactions 0x80 = All messages 0xC0 = Full trace	Reports accesses to the Queue file.

Table 5. Mask Values for DEBUG Parameter 2

Category	Mask	Description
Global diagnostics	0x0100	Enables error messages from the card and from failed faxes. Shows sent and received CSID values, as well as the phone number dialed.
T.30 diagnostics	0x0400	Enables all the above plus DIS bits and DCS bits sent and received. Also logs messages about image compressions and conversions written to disk.
Total diagnostics	0x0800	Enables all the above plus information in the queue record before being processed. Also logs information on the file and image being sent.

Table 6. Mask Values for SRAMMask and LogFileMask

Mnemonic	Bit	Description
DMASK_ERROR	1	A critical error occurred either at FAX or PRI level
DMASK_DIALERROR	1	Failures on Dialogic functions
DMASK_GENERAL	2	General Information (e.g. Number of channels, Version numbers)

Mnemonic	Bit	Description
DMASK_REGISTRY	3	Registry (read / write) related activity
DMASK_ERR_REGISTRY	3	Registry error related info
DMASK_GLCOMMAND	4	GDK related commands (e.g. GFXSHUTDOWN, GFXRECORD, …)
DMASK_GLQSUBMQREC	5	On-hold, Free related activity
DMASK_GLPIPEOP	6	Pipe related activity
DMASK_GLMSG	7	Device related activity
DMASK_SCBUS	8	SCbus related activity
DMASK_SCMANAG	9	Resource management activity
DMASK_DIALMSG	10	Dialogic General information (ISDN frame, D-Channel, …)
DMASK_DIALCMD	11	Dialogic commands related information
DMASK_SPECIAL	12	Dialogic special functions
DMASK_DIALEVENT	13	ISDN Event related information (all ccev events)
DMASK_DIALCC	14	Call control related information (all cc commands)
DMASK_DIALINFO	15	Dialogic / GDK interaction and board handles info.
DMASK_FIRMWARE	16	Debug information specific to Firmware

Mnemonic	Bit	Description
DMASK_DISPATCHER	17	Debug information specific to Dispatcher

NOTE: There are 32 possible Debug Levels. Undefined Debug levels are reserved for future use.

Enabling Transparent PRI Debug

When Transparent PRI mode is enabled in the GDK System Service, the UseSRAM command is automatically enabled. Trace information about the ISDN and FAX resources is provided during system startup and initialization (this information is not outputted in non-ISDN mode).

In addition, the Dialogic ISDN cc_Trace logging mechanism is available by setting the following commands:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Dialogic\Gammalink\Debug

        TraceTrunkNumber

        TraceFileName
```

ISDN trace is enabled by setting TraceFileName to a full path and filename and by setting TraceTrunkNumber to a valid ISDN trunk. The resultant file from the TraceFileName command can be used as input to Dialogic's PRITRACE tool.

Debug Command Examples

To obtain debugging information for two categories, use the logical OR to combine the hex values. For example, to get polling diagnostics for type only, the first four digits of the mask are 0x04. Then, to get polling diagnostics for both type and parameters, the mask is 0x08. A full trace of polling diagnostics would be as follows:

```
0x04 OR 0x08 = 0x0C
```

where: the first byte is to file and the second byte is to screen.

Logging of general trace information and serious error messages can be enabled with the following parameter:

```
DEBUG 0x0101
```

The recommended debug level for most diagnostic purposes is the following mask value:

```
DEBUG 0xb7b7
```

To obtain additional fax protocol diagnostics, also set the second DEBUG parameter:

```
DEBUG 0xb7b7 0xc0c0
```

Debug Setting Dependencies

Two parameters have been added that control how the Debug information is logged:

- UseSRAM
- DebugToSRAM

The UseSRAM parameter lets you specify whether to log Debug levels with a bit setting of 1 to the Shared Memory. If the parameter is set to “yes” only the debug levels having a corresponding mask set to 1 in the bit mask SRAMMask are passed to Shared Memory.

The DebugToSRAM parameter lets you specify whether to send debug information with Debug level DMASK_FIRMWARE if the information is originated by the firmware (XXX_put), or DMASK_DISPATCHER if the information is originated by the dispatcher to Shared Memory.

For more information about these parameters, refer to the New Parameters section in Chapter 3.

3. Configuration Commands

Not all commands are available with every CP Fax board. For the capabilities available on the boards installed in the host system, see the appropriate hardware installation guide for the specific board.

The Dispatcher and the firmware may be customized with commands in the registry. The GDK configuration commands used in the registry and their functions are summarized in Table 7. The remainder of this chapter is an alphabetical list of configuration commands.

Configuration Commands Summary

NOTE: In the following table, the term “on-the-fly” refers to files converted automatically at the time of transmission on the CP Fax board.

Table 7. Summary of Configuration Commands

Command	Function
BUFFERS	Sets the maximum number of queue record buffers.
CHANNELID	Defines logical-to-physical mapping of a fax channel.
CHASSIS	Defines the number of the host machine. (always 1)
CONTROLT	Sets the interval for checking the Control List.
COUNTRY	Defines the country code for each fax channel.
CSID	Sets the CSID for a fax channel.
DEBUG	Sets the debug level.
FIRMWARE	Downloads the onboard software for a fax channel.
GFCCONTROL 36	Specifies that all pages of a multi-page fax be saved into one, multi-page TIFF file.
GFCCONTROL 37	Sets up flexible naming, such that the file name provided by the application is the one used.
GFXBOTTOMMARGIN	Sets the bottom margin of the fax page for on-the-fly ASCII conversions.

GFXCARRYON	Sets the number of unacceptable pages sent before termination.
GFXCHARSET	Sets the font slot for the default on-the-fly ASCII conversions.
GFXDID	Specifies the trunk type connected to a fax channel.
GFXDIGITS	Sets the number of digits to be received.
GFXDTMFTIMEOUT	Sets timeout interval when waiting for digit input.
GFXDTMFTONE	Specifies the tone to issue when answering a call.
GFXECM	Sets the error correction mode.
GFXEXTEND	Selects the ASCII character set used for on-the-fly ASCII conversions.
GFXFAXCONTROL 29 GFXFAXCONTROL 28	Both commands, in order, allow GDK to collect DTMF digits immediately after going off-hook on an analog line.
GFXFAXCONTROL 40 GFXFAXCONTROL 41 GFXFAXCONTROL 42 GFXFAXCONTROL 43 GFXFAXCONTROL 44	The function controls GFXFAXCONTROL 40 through GFXFAXCONTROL 44 enable the board to send two frequencies at one time, for a given duration.
GFXFAXCONTROL 71	Enables T.30 subaddressing, which allows a string of characters and numbers to be sent with the fax.
GFXFAXCONTROL 72	Changes the wink duration when using GFXDID set to 1 or 6 for DID wink or T1 wink respectively.
GFXFAXCONTROL 73	Specifies the T.30 subaddressing (sending) field delimiter. Used in conjunction with GFXFAXCONTROL 71.
GFXFAXCONTROL 74	Turns off T.30 password collection if it is not needed by the application.
GFXFAXCONTROL 1020	Allows the reception of Binary File Transfer (BFT) between other systems that support the transmission of BFT.
GFXFAXCONTROL 1021	Enables the transmission of BFT transactions to other systems capable of receiving BFT.
GFXFINE	Accepts only fine resolution incoming faxes.

GFXFORM	Selects the image format of a received fax and sanitization of faulty scanlines.
GFXHEADER	Customizes the header on outgoing faxes.
GFXLEFTMARGIN	Sets the left margin of the page for on-the-fly ASCII conversions.
GFXOVERLAY	Controls the overlay header feature.
GFXPAGELENGTH	Sets the number of character lines per page for on-the-fly ASCII conversions.
GFXRECM	Controls receive ECM.
GFXRECVPATH	Sets the directory path used for a fax channel to receive faxes.
GFXREJBURST	Sets maximum number of consecutive bad scanlines that will be tolerated before a page is rejected.
GFXREJCOUNT	Sets absolute number of bad scanlines that will be tolerated before a page is rejected.
GFXREJPERCENT	Sets percent of bad scanlines that will be tolerated before a page is rejected.
GFXRIGHTMARGIN	Sets the right margin of the page in characters for on-the-fly ASCII conversions.
GFXRLENGTH	Selects the page length of a received fax.
GFXRTNRETRAIN	Specifies action to be taken when an illegible page has been sent.
GFXRTPRETRAIN	Specifies action to be taken when requiring a higher speed or change in resolution.
GFXRT6	Controls reception of T.6 encoding.
GFXRTWOD	Controls reception of Group 3 2-D line compression.
GFXRWIDTH	Selects the page width of a received fax.
GFXSCANTIME	Sets the scanline timing for incoming calls.
GFXSECM	Controls send ECM.
GFXSHUTDOWN	Specifies fax channel state, such as Dial only or Answer only.
GFXSPEAKER	Controls the speaker.

GFXST6	Controls transmission of T.6 encoding.
GFXSTWOD	Controls transmission of Group 3 2-D line compression.
GFXTOPMARGIN	Sets the top margin of the fax page for on-the-fly ASCII conversions.
GFXWAIT	Sets the wait-for-answer time in seconds for a fax channel.
INIT	Initializes the fax channel.
LOADFONT	Downloads a font for on-the-fly ASCII conversions.
MODEMCTRL 1024	Changes the default dialing type when placing a call.
MODEMCTRL 2054	Sets the number of rings before answering.
MODEMCTRL 2066	The dial string may contain the character ";" to indicate a pause while dialing. This command sets the time value for this character.
NUMCHAN	Defines number of fax channels in the system.
QUEUET	Sets the time between Dispatcher checks of the Pending List.
STATUST	Sets the interval between writes of status table to disk.
UPDATET	Sets the maximum time between writes to the log file.

Configuration Commands

Brief descriptions of the configuration commands used by the software are listed alphabetically on the following pages. In these descriptions, arguments that appear in brackets ([]) are optional.

BUFFERS

DESCRIPTION	Specifies the maximum number of queue-record buffers to allocate for the Dispatcher buffer pool.
SYNTAX	BUFFERS <n>
VALUE	n A value twice the number of channels.

CHANNELID

DESCRIPTION	Defines the logical-to-physical mapping for each fax channel in the system and, optionally, gives the channel a symbolic name or a "modem_id." A modem_id in the form "GFAX<n>.<m>." <n> The number of the chassis (see CHASSIS <n>). <m> The channel number. CHANNELID <string> <n>

CHASSIS

DESCRIPTION	Defines the number of the chassis on which the Dispatcher is executing.
SYNTAX	CHASSIS <n>
VALUE	n The chassis number. <n> = 1 is the only value currently supported.

CONTROLT

DESCRIPTION	Specifies the time in seconds between Dispatcher checks of the Control List.
SYNTAX	CONTROLT <n>
VALUE	n A number of seconds. Reasonable numbers are between 30 seconds and 60 seconds. The default is <n> = 60.

COUNTRY

DESCRIPTION	Defines the country code, which is used for the country-specific modem operating parameters.	
SYNTAX	COUNTRY <n>	
VALUE	n	This code must be specified for the fax channel to function correctly. The country code for the United States is 1. There is no default value. For a list of country codes, see Appendix D. Refer to the "Setting the Country Code Properly" section.

CSID

DESCRIPTION	Specifies the default customer subscriber identification (CSID) number for each fax channel.	
SYNTAX	CSID <phonenumber>	
VALUE	phonenumber	For a host system with multiple outgoing fax lines, the CSID should reflect the phone number of an incoming fax line. If this command is absent, a blank CSID is transmitted. The ITU recommends that the CSID be set to the international phone number of the fax channel. To ensure the greatest compatibility with remote fax machines, do not include spaces or alpha characters in the phonenumber parameter. The maximum length of the CSID is 20 characters.

DEBUG

DESCRIPTION	Sets the Debug mask.	
SYNTAX	DEBUG <n>	
VALUE	n	Refer to the tables in the Using the Debug Masks section of Chapter 2.

FIRMWARE

DESCRIPTION	Downloads the on-board software so that the channel can be used.
SYNTAX	FIRMWARE C:\FAX\GFXCX.BIN

GFCCONTROL 36

DESCRIPTION	This command specifies that all pages of a multi-page fax be saved into one, multi-page TIFF file. This command changes the GDK default, which is that each page of a multi-page fax received is saved into a separate file.	
SYNTAX	GFCCONTROL 36 <n>	
VALUE	n	Is one of the following values: 0 All pages are received into one file. 1 Each page is stored in a separate file. (Default) n Stores a specified number of pages per file, and then opens a new file. Use a value other than 1.

GFCCONTROL 37

DESCRIPTION

By default, GDK generates a new filename for each received fax. This filename is generated by using the channel, fax job number, and page number. This command sets up flexible naming, such that the file name provided by the application is the one used. This only works if:

- Multi-page receive is enabled
- An application is running to provide its own unique received file name

This command also sets a page flush in multi-page TIFF receive, meaning that each successfully received page is flushed to disk during the page break.

SYNTAX

GFCCONTROL 37 <n>

VALUE

n — Is one of the following values:

0 Specifies default page naming. Use this setting with single page TIFF receive. (Default)

1 Specifies to flush each page to disk received. If an error occurs, all pages are saved.

2 Enables flexible naming convention.

3 Specifies 1 and 2 together. This option is recommended for multi-page TIFF receive when using GFCCONTROL 36 0 or 1.

GFXBOTTOMMARGIN

DESCRIPTION

Sets the number of text lines from the bottom of the page. Used for the bottom margin of a fax page for on-the-fly ASCII conversions.

SYNTAX

GFXBOTTOMMARGIN <n>

VALUE

n — Specifies the number of lines. The range is 0 to 65 lines; the default is 3.

GFXCARRYON

DESCRIPTION	Specifies the number of unacceptable pages that may be sent before the call is terminated. A remote fax machine may send a RetrainNegative (RTN) signal between pages to indicate that part of the transmission it is receiving is faulty. GDK continues to send pages even after receiving an RTN. The GFXCARRYON command permits a call to be terminated, so a "retraining" command can take control when the remote machine rejects a page.
SYNTAX	GFXCARRYON <n>
VALUE	n Is one of the following: 0 Is an infinite number of pages. The default value is 0. n Is the number of RTN before terminating the call.
SEE ALSO	GFXRTNRETRAIN

GFXCHARSET

DESCRIPTION	Selects the font style used for on-the-fly ASCII conversions.
SYNTAX	GFXCHARSET <n>
VALUE	n Is one of the following values: 0 System font (used for page headers) 1 Standard font — Sans Serif 12 pt. (Default) 2 Auxiliary font — Courier 12 pt. 3 Compressible font (line printer)
SEE ALSO	LOADFONT, GFXEXTEND

GFXDID

DESCRIPTION	Specifies the signaling and digit type (DTMF or pulse) for a given channel.	
SYNTAX	GFXDID <did_type> <digit_type>	
VALUE	did_type	Is one of the following values: 0 Analog loop start (Default) 1 DID wink-start 2 Reserved for future use 3 Reserved for future use 4 Reserved for future use 5 Reserved for future use 6 T.1 wink start
	digit_type	Is one of the following values. 0 DTMF (Touch-Tone) 1 Reserved 2 Loop pulse
EXAMPLE	This example is for a system with one CPD board: GFXDID 1 0 This instructs the channel to receive DTMF digits and have a DID wink-start line interface.	

GFXDIGITS

DESCRIPTION	Sets the number of digits to be received when answering a call. The terminating digit sends a signal to the channel that input is finished.	
SYNTAX	GFXDIGITS <quantity> <terminating>	
VALUE	quantity	Specifies the number of digits. Enter the maximum number of digits to process, including any terminating digits. Terminating digits are only used if the digits received may vary in length.
	terminating	To specify the terminating digit, enter the parameter for the specific terminating digit. Use "0" to indicate no terminating digit. The acceptable parameter range for terminating digits is 0 to 65,535. The terminating digits and their parameters are listed in Table 8.
EXAMPLE	This command is an example of using the "#" character to terminate input of a string of four digits: GFXDIGITS 5 2048 where "5" designates the maximum number of digits to expect. The "2048" signifies that the "#" character is desired as a terminator. Multiple terminating digits also can be used. For example, if "*," "7," or "1" are the terminating digits, the number entered would be 1089, which is the sum of the parameters, that is, 1024 + 64 + 1 = 1089.	
SEE ALSO	GFXDTMFTIMEOUT, GFXDTMFTONE	

Table 8. Terminating Digit Parameters

Digits	Parameter
1	1
2	2
3	4
4	8
5	16
6	32
7	64
8	128
9	256
0	512
* (star)	1024
# (number sign)	2048
A	4096
B	8192
C	16384
D	32768

GFXDTMFTIMEOUT

DESCRIPTION	Sets the timeout interval when waiting for DTMF input. A timeout for DTMF input must be established or the channel will not wait for any digits to be inputted.	
SYNTAX	GFXDTMFTIMEOUT <interdigit_timeout> <total_timeout>	
VALUE	interdigit_timeout	Is the waiting time between digits. The minimum value is one second, and any value between 1 and 32,767 is valid.
	total_timeout	Is the length of the wait to receive all digits. The minimum value is one second, and any value between 1 and 32,767 is valid.
EXAMPLE	For a three-digit DTMF input, this command is recommended: GFXDTMFTIMEOUT 5 10 This instructs the channel to wait no more than five seconds between each digit it receives, and to wait no more than ten seconds to receive all the digits. If only three digits are received, a five-second timeout between digits is ample. For more than a three-digit input, the total timeout may have to be increased, for example, by one second per digit over three.	
SEE ALSO	GFXDTMFTONE, GFXDIGITS	

GFXDTMFTONE

DESCRIPTION	Specifies the tone to issue when answering a call. This feature is useful for unattended DTMF (dual-tone multi-frequency) operation. This command is used to prompt a calling party to enter tones. This command only emits a single frequency tone. If a true DTMF tone is needed, please refer to the GFXFAX commands. This command is not used with DID trunk interface boards, such as the CPD or CPD/220.	
SYNTAX	GFXDTMFTONE <frequency> <duration>	
VALUE	frequency	Specify a value in Hertz, up to 3000.
	duration	Enter, in milliseconds, the length of time for the tone to sound.
EXAMPLE	These are the recommended parameters: GFXDTMFTONE 440 1000 With this command, the channel sounds the musical note "A" (440 Hz) for one second (1000 milliseconds). That is, after the channel has gone off-hook, it sends one "A" note for one second to prompt for tones from the calling party.	
SEE ALSO	GFXDTMFTIMEOUT, GFXDIGITS	

GFXECM

DESCRIPTION	Sets the Error Correction Mode (ECM), which can be set to operate in three modes.	
SYNTAX	GFXECM <send> <receive>	
VALUE	send	Is one of the following settings: 0 Turned off so the ECM feature is not used. (Default) 1 Send from any ECM-capable fax machine or board. 2 Send from any ECM-capable fax machine or board only.
	receive	Is one of the following settings: 0 Turned off so the ECM feature is not used. (Default) 1 Receives from any ECM-capable fax machine or board. 2 Receives from any ECM-capable fax machine or board only, and rejects all other transmissions.
EXAMPLE	For example, to instruct board one to send and receive in the ECM mode whenever possible, use this command: GFXECM 1 1 To turn it off, use this command: GFXECM 0 0	

GFXEXTEND

DESCRIPTION Selects the ASCII character subset used for on-the-fly ASCII conversion. Any character codes outside the specified range are ignored.

SYNTAX GFXEXTEND <n>

VALUE n Is one of the following values:

- 0 Character set 32 to 127ASCII text only. These are the alphabetic characters used for composition in English. (Default)
- 1 Character set 32 to 255 extended ASCII text only.
- 2 Character set 0 to 255 full ASCII text.

GFXFAXCONTROL 29
GFXFAXCONTROL 28

DESCRIPTION In an analog environment, digits are able to be collected two seconds after the channel goes off-hook. For digits that are being passed automatically from a switch, this time is too long. These commands — in this order — allow GDK to collect DTMF digits immediately after going off-hook on an analog line.

SYNTAX GFXFAXCONTROL 29 2000

GFXFAXCONTROL 28 0

NOTE: The GFXFAXCONTROL 28 and 29 commands must be used together in order to provide the expected results.

GFXFAXCONTROL 71

DESCRIPTION	This command enables T.30 subaddressing, which allows a string of characters and numbers to be sent with the fax. This string is known as the subaddress. It allows fax servers to do routing based on this subaddress.	
SYNTAX	GFXFAXCONTROL 71 <n>	
VALUE	n	Indicates whether T.30 subaddressing is enabled. Specifying a default value of 0 disables the T.30 subaddressing option and a value of 1 enables the option.

GFXFAXCONTROL 72

DESCRIPTION	This command changes the wink duration when using GFXDID set to 1 or 6 for DID wink or T1 wink respectively. Adding this command unnecessarily can cause communication problems between the CP Fax board and the service provider.	
SYNTAX	GFXFAXCONTROL 72 <n>	
VALUE	n	Is the number of milliseconds recommended by the service provider. The default is 150 milliseconds. Most installations operate well using the default.

GFXFAXCONTROL 73

DESCRIPTION	This command is used in conjunction with the GFXFAXCONTROL 71 command. When using GFXFAXCONTROL 71, use GFXFAXCONTROL 73 to specify the T.30 subaddressing (sending) field delimiter.	
SYNTAX	GFXFAXCONTROL 73 <n>	
VALUE	n	Is one of the following values:
	2	Is "
	3	Is # (Default)
	4	Is $
	5	Is %
	6	Is &

GFXFAXCONTROL 74

DESCRIPTION	If a T.30 password is sent, GDK collects it and places it in the user_id field for routing purposes. The GFXFAXCONTROL 74 command turns off T.30 password collection if it is not needed by the application.	
SYNTAX	GFXFAXCONTROL 74 <n>	
VALUE	n	Is one of the following values:
	0	Disables password collection
	2	Enables password collection (Default)

GFXFAXCONTROL 1020

DESCRIPTION	Allows the reception of Binary File Transfer (BFT) between other systems that support the transmission of BFT. The only field in the BFT header that is honored is the filename. All other fields are ignored and discarded. The file is stored in the %GFAXR% directory with the same name as the send file, if possible.		
SYNTAX	GFXFAXCONTROL 1020 <n>		
VALUE	n	Is one of the following values:	
		0	Disables T.434 BFT reception (Default)
		1	Enables T.434 BFT reception

GFXFAXCONTROL 1021

DESCRIPTION	Enables the transmission of Binary File Transfer (BFT) transactions to other systems capable of receiving BFT. BFT (T.434) is disabled by default. ECM must also be enabled for BFT transmission.		
SYNTAX	GFXFAXCONTROL 1021 <n>		
VALUE	n	Is one of the following values:	
		0	Disables all BFT transmission (Default)
		1	Not supported
		2	Enables T.434 BFT transmission

GFXFINE

DESCRIPTION	Accepts incoming faxes in fine-resolution mode only and fails all calls that are not in fine-resolution mode.
SYNTAX	GFXFINE <n>
VALUE	n Is one of the following values: 0 Turns off receiving of fine-only resolution faxes. (Default) 1 Turns on receiving of fine-only resolution faxes. Rejects incoming faxes in standard resolution.

GFXFORM

DESCRIPTION	Accepts incoming faxes in fine-resolution mode only and fails all calls that are not in fine-resolution mode.
SYNTAX	GFXFORM <n>
VALUE	n Is one of the following values: 0 Same format that is used for transmission (Default) 3 TIFF Type 3 1-D; with sanitization 4 TIFF Type 3 2-D 5 TIFF Type 4
EXAMPLE	GFXFORM 3 If n is set to 0, no value is set. The image is written to disk in the same format it is received. For example, if the file is transmitted in TIFF Type 3 2-D format, it is received and written to disk as TIFF Type 3 2-D.

GFXHEADER

DESCRIPTION	Customizes the header on outgoing faxes to contain user-defined information.	
SYNTAX	GFXHEADER <format>	
VALUE	format	Is information to be replaced. Table 9 lists the variables, the field widths, and the text to be replaced. Up to 95 characters can be printed on the header.
EXAMPLE	For example, a customized header written with these variables: &day&abmon&year &12hr:&min&ampm From:&from To:&to Page &page would appear as follows: 25Aug97 1:30pm From:408-969-5200 To:1-203-359-9203 Page 1 To display a header, the header bit must be set in the queue record. To turn on the header in a queue record, see the transmit_control field description in Chapter 4.	

Table 9. Variables, Field Widths, and Text

Variable	Field Width	Text
&header	20	The header field from the queue record.
&user	32	The user field of the current queue record.
&from	20	The sent CSID; that is, the CSID from which the fax is sent.
&to	20	The receive CSID; that is, the CSID to which the fax is sent. &to is right-justified.
&phone	20	The phone_no field of the queue record.
&page	3	The current page number in decimal number format (1 to 999). The number is right justified; the field is zero filled (001).

Variable	Field Width	Text
&time	9	The current time in country-dependent format. In the U.S.A., the format is “hh:mm:ss”; in other countries, the format may vary.
&12hr	2	The hour based on a 12-hour clock in decimal-number format (01 to 12).
&24hr	2	The hour based on a 24-hour clock in decimal-number format (00 to 23).
&ampm	2	Designation to indicate before or after noon. The values are “AM” and “PM”.
&min	2	The minute in decimal-number format (00 to 59).
&sec	2	The second in decimal-number format (00 to 59).
&date	9	The date in country-dependent format. In the U.S.A., the format is “mm/dd/yy”; in other countries, the format may vary.
&abmon	3	An abbreviation of the month. The values are “Jan”, “Feb”, “Mar”, “Apr”, “May”, “Jun”, “Jul”, “Aug”, “Sep”, “Oct”, “Nov”, and “Dec”, which are unchangeable.
&month	2	The month in decimal-number format (01 to 12).
&strmon	10	The month of the year. The values are “January”, “February”, “March”, “April”, “May”, “June”, “July”, “August”, “September”, “October”, “November”, and “December”, which are unchangeable.
&day	2	The day of the month in decimal-number format (01 to 31).
&year	2	The year without century in decimal-number format (00 to 99).
&lyear	4	The year with century in decimal-number format (0000 to 9999).

GFXLEFTMARGIN

DESCRIPTION	This command sets the left margin of the page in characters for on-the-fly ASCII conversion.
SYNTAX	GFXLEFTMARGIN <n>
VALUE	n Is a value in the range 0 to 215. The default is 14.

GFXPAGELENGTH

DESCRIPTION	Sets the number of lines per page used for on-the-fly ASCII conversion.
SYNTAX	GFXPAGELENGTH <n>
VALUE	n Is the number of text lines per page. The default is n = 66. Use n = 0 for variable-length documents in which the ASCII file equals one fax page.
	NOTE: When converting ASCII files on the channel, to accommodate more lines per page for the A4 paper size, add the following line to the registry: GFXPAGELENGTH <n> Where n is the number of text lines per page.

GFXRECM

DESCRIPTION	Controls the Error Correction Mode (ECM) for incoming faxes.
SYNTAX	GFXRECM <n>
VALUE	n Is one of the following values: 0 Turns off the ECM feature. (Default) 1 Receives from any ECM-capable fax machine or board. 2 Receives only ECM transmissions, and rejects all other transmissions.
EXAMPLE	For example, to instruct the channel to receive in the ECM mode whenever possible, use this command: GFXRECM 1

GFXRECVPATH

DESCRIPTION	Sets the fully-qualified path and filename template used for a fax channel to receive faxes. This command defines both the directory in which received files will be placed and the filename template to be used.	
SYNTAX	GFXRECVPATH <path> <filename>	
VALUE	path	Is the path to the directory.
	filename	Is the filename template for the files that are received.
EXAMPLE	For example, the following command could be used to put all the received files into the \RECVD\BD1 directory on the E: drive:	
	GFXRECVPATH E:\RECVD\BD1\F001P001.TIF	

GFXREJBURST

DESCRIPTION	Sets the maximum number of consecutive bad scanlines that are tolerated before a page is rejected. This helps to control the quality of incoming faxes. This command works only if sanitization is enabled with the GFXFORM command.	
SYNTAX	GFXREJBURST <n>	
VALUE	n	Specifies the number of allowable, consecutive bad scanlines. By default, this value is -1 (off).

GFXREJCOUNT

DESCRIPTION	Sets the total number of bad scanlines that are tolerated before a page is rejected. This helps to control the quality of incoming faxes. This command works only if sanitization is enabled with the GFXFORM command.	
SYNTAX	GFXREJCOUNT <n>	
VALUE	n	Specifies the total number of allowable bad scanlines. By default, this value is -1 (off). The range is 0 to 32,767.

GFXREJPERCENT

DESCRIPTION	Sets the percentage of bad scanlines that are tolerated before a page is rejected. This helps to control the quality of incoming faxes. It specifies the number of bad scanlines per 100 scanlines. This command works only if sanitization is enabled with the GFXFORM command.	
SYNTAX	GFXREJPERCENT <n>	
VALUE	n	Specifies the total percent of allowable bad scanlines. By default, this value is -1 (off). The range is 0 to 100.

GFXRIGHTMARGIN

DESCRIPTION	Sets the right margin of the page in characters, relative to the left edge, for on-the-fly ASCII conversions.	
SYNTAX	GFXRIGHTMARGIN <n>	
VALUE	n	Specifies the right margin of the page in characters. The default is 94. The range is 0 to 215.

GFXRLENGTH

DESCRIPTION	Selects the page length of incoming faxes.
SYNTAX	GFXRLENGTH <n>
VALUE	n Is one of the following values: 0 A4 size 1 A4 and B4 size 2 Unlimited length (Default) This is a request to the sending fax machine. If the remote machine still sends a page longer than A4, the transmission will not fail. The image information will be written to disk along the length of the transmission.
SEE ALSO	GFXRWIDTH

GFXRTNRETRAIN

DESCRIPTION	Selects an action to be taken when an illegible page has been sent. The valid rates are the transmission rates for the modem, such as 14400.
SYNTAX	GFXRTNRETRAIN <n>
VALUE	n Is one of the following values: 0 Dispatcher resumes transmission at the current rate. (Default) <0 Retraining starts at the next lower rate. >0 Contains the retraining rate.
SEE ALSO	GFXMAXRATE

GFXRTPRETRAIN

DESCRIPTION	Specifies an action to retrain (in hopes of a higher speed or change in resolution) for pages received correctly.	
SYNTAX	GFXRTPRETRAIN <n>	
VALUE	n	Is one of the following values:
		0 Dispatcher resumes transmission at the current rate. (Default)
		<0 Retraining starts at the next lower rate.
		>0 Contains the retraining rate.

GFXRT6

DESCRIPTION	Controls the reception of T.6 encoding. T.6 encoding requires Error Correction Mode (ECM). ECM must be enabled to receive any T.6 encoded faxes.	
SYNTAX	GFXRT6 <n>	
VALUE	n	Is one of the following values:
		0 Turns off; cannot receive T.6 encoding. (Default)
		1 Receives T.6 encoding if sender has capability.
		2 Receives only T.6 encoding; refuses all others.

GFXRTWOD

DESCRIPTION	Controls TIFF Type 3 2-D line or transmission compression of received faxes, and writes a 2-D image to disk unless otherwise set by GFXFORM.
SYNTAX	GFXRTWOD <n>
VALUE	n Is one of the following values: 0 Turns off; cannot receive TIFF Type 3 2-D compression. 1 Receives TIFF Type 3 2-D compression if sender has capability. (Default) 2 Receives TIFF Type 3 2-D compression; refuse all other formats.

GFXRWIDTH

DESCRIPTION	Selects the page width of incoming faxes.
SYNTAX	GFXRWIDTH <n>
VALUE	n Is one of the following values: 0 A4 size (Default) 1 A4 and B4 size 2 Any width
SEE ALSO	GFXRLENGTH

GFXSCANTIME

DESCRIPTION	Sets the scanline timing in milliseconds for incoming calls. When receiving from a remote fax machine, GDK specifies in the DIS field that it is capable of accepting scanlines at a certain rate.	
SYNTAX	GFXSCANTIME <n>	
VALUE	n	Is one of the following values:
	0	0-msec fax machine (Default)
	5	5-msec fax machine
	10	10-msec fax machine
	20	20-msec fax machine
	40	40-msec fax machine

GFXSECM

DESCRIPTION	Controls the ECM for outgoing faxes.	
SYNTAX	GFXSECM <n>	
VALUE	n	Is one of the following values:
	0	Turns off so the ECM feature is not used. (Default)
	1	Sends to any ECM-capable fax machine or board.
	2	Sends to an ECM-capable fax machine or board only, and fails all other transmissions.
EXAMPLE	To instruct board one to send in the ECM mode whenever possible, enter the following:	
	GFXSECM 1	

GFXSHUTDOWN

DESCRIPTION Specifies the state of each fax channel.

SYNTAX GFXSHUTDOWN <n>

VALUE n Is one of the following values:

0 Answers incoming calls and checks for Pending queue requests.

or one or a combination of the following bit masks:

1 Ignores future incoming calls. For Dial-only operation.

2 Ignores future Pending queue requests. For Answer-only operation.

3 Does not send or receive.

4 Busies out the line; that is, makes the phone line busy by staying off- hook.

The GFXSHUTDOWN values 0 and 2 are also controlled by the AutoReceive registry flag configuration.

If the AutoReceive registry flag is set to “yes” when the GFXSHUTDOWN value is set to 0 or 2, the channels receive incoming faxes automatically. The associated queue record is stored in the queue file. This occurs even if there is no application controlling that channel.

If the AutoReceive registry flag is set to “no” when the GFXSHUTDOWN value is set to 0 or 2, the channels do not receive incoming faxes when there is no application using the channel. The transmission fails and the sender receives an error indicated that the fax was not received.

The default setting for the AutoReceive registry flag is “no”.

EXAMPLE These control bits may be combined. Thus, if n = 3, the second and third bits are in effect; if n = 7, the second, third, and fourth bits are in effect. The value of n can also be 5 or 6.

GFXSPEAKER

DESCRIPTION	Disables and enables the speaker on the CP Fax board. Not all CP fax boards have speakers. See the hardware manual for the board installed in the host system to determine whether or not the board has a speaker.
SYNTAX	GFXSPEAKER <n>
VALUE	n Is one of the following values: 0 No speaker. (Default) 1 Enables speaker during wait for answer tone. 2 Enables speaker during dial tone dialing. 3 Enables speaker continuously.

GFXST6

DESCRIPTION	Controls the transmission of T.6 encoding. T.6 encoding requires ECM. Therefore, ECM must be turned on first before the T.6 setting, or the firmware will reject the T.6 transmission.
SYNTAX	GFXST6 <n>
VALUE	n Is one of the following values: 0 Turns off; cannot send T.6 encoding. (Default) 1 Sends T.6 encoding if recipient has capability. 2 Sends only T.6 encoding; fails transmission if recipient does not have capability.

GFXSTWOD

DESCRIPTION	Controls TIFF Type 3 2-D line compression of fax transmissions.	
SYNTAX	GFXSTWOD <n>	
VALUE	n	Is one of the following values:
	0	Turns off; cannot send TIFF Type 3 2-D compression.
	1	Sends TIFF Type 3 2-D compression if recipient has capability. (Default)
	2	Sends TIFF Type 3 2-D compression; fails transmission if recipient does not have capability.

GFXTOPMARGIN

DESCRIPTION	Sets the top margin of the fax page in text lines for on-the-fly ASCII conversions.	
SYNTAX	GFXTOPMARGIN <n>	
VALUE	n	Is a value from 0 to 65. The default is 3.

GFXWAIT

DESCRIPTION	Sets the wait-for-answer time in seconds for each fax channel; that is, the length of time the channel will stay idle waiting for an incoming call to arrive before searching for something to send in the default answer state.	
SYNTAX	GFXWAIT <n>	
VALUE	n	Is time in seconds.

INIT

DESCRIPTION	Initializes the specified fax channel. Each channel must be initialized with INIT before it can be used.
SYNTAX	INIT

LOADFONT

DESCRIPTION	Downloads a font into the one of the font slots used for on-the-fly ASCII conversions. The GFXCHARSET command is used to select the font number.	
SYNTAX	LOADFONT <n> <filename>	
VALUE	n	Is a font number, from 0 to 3.
	filename	Is the file containing the font.

MODEMCTRL 1024

DESCRIPTION	Changes the default dialing type when placing a call. The default is dependent upon the country code and version of the software that you are using. In the United States and Canada, pulse is the default setting for GDK 5.0 and earlier. Tone is the default setting for GDK 4.0 or later.	
SYNTAX	MODEMCTRL 1024 <n>	
VALUE	n	Is one of the following values:
	0	Is for pulse dialing
	1	Is for tone dialing

MODEMCTRL 2054

DESCRIPTION	Sets the number of rings before answering.	
SYNTAX	MODEMCTRL 2054 <n>	
VALUE	n	Is the number of rings. The default is 1.

MODEMCTRL 2066

DESCRIPTION	The dial string may contain the character ";" to indicate a pause while dialing. This command sets the time value for the ";" character.	
SYNTAX	MODEMCTRL 2066 <n>	
VALUE	n	Is the time in milliseconds. The default is 1500 milliseconds.

NUMCHAN

DESCRIPTION	Defines the number of fax channels in the system. This must be defined before the channels can be addressed.	
SYNTAX	NUMCHAN <n>	
VALUE	n	Is the number of fax channels in the system.

QUEUET

DESCRIPTION	Sets the time in seconds and milliseconds between Dispatcher checks of the Pending List.	
SYNTAX	QUEUET <m> <n>	
VALUE	m	Is the time in seconds. The range is 0 to 32,767 seconds; reasonable settings are between 5 and 120. The default is 45.
	n	Is the time in milliseconds. The default is 0.

STATUST

DESCRIPTION	Sets the interval between writes of the status table to the status file on disk.	
SYNTAX	STATUST <m> <n>	
VALUE	m	Is the time in seconds. The default is 0. The range is 0 to 32,767 seconds.
	n	Is the time in milliseconds. The default is 0. A reasonable number is 5.

UPDATET

DESCRIPTION	Sets the maximum time in seconds and milliseconds between writes to the log file. The log file may be written more frequently if significant transactions take place.	
SYNTAX	UPDATET <m> <n>	
VALUE	m	Is the time in seconds. The range is 0 to 32,767 seconds. The default is 300. Reasonable numbers are between 15 seconds and 600 seconds.
	n	Is the time in milliseconds. The default is 0.

New Parameter Summary

The new parameters that have been added are grouped into four categories:

- Management
- ISDN
- ErrorMapping
- Debug

Each parameter category has a specific registry key path.

- Management
  KEY_LOCAL_MACHINE\SOFTWARE\Dialogic\Gammalink\Management
- ISDN

  HKEY_LOCAL_MACHINE\SOFTWARE\Dialogic\Gammalink\ISDN
- ErrorMapping

  HKEY_LOCAL_MACHINE\SOFTWARE\Dialogic\Gammalink\ErrorMapping
- Debug

  HKEY_LOCAL_MACHINE\SOFTWARE\Dialogic\Gammalink\Debug

To modify the default settings of the any of the parameters, use a registry editing tool.

CAUTION

Using a registry editor application incorrectly can cause serious, system-wide problems that may require you to reinstall Windows to correct them. Dialogic cannot guarantee that any problems resulting from the use of a registry editor can be solved. Use these tools at your own risk.

The following section contains summary tables and parameter definitions for each category of the new parameters.

Management Parameters

These parameters control the management of the fax and ISDN features.

Table 10. Management Parameters

Parameter	Description
PRILayerEnable	Specifies whether to use the "Transparent PRI Support" or not.
NumberOfTrunks	Indicates the number of PRI trunks in the chassis.
ChannelsPerTrunk	Indicates the number of ISDN channels per PRI trunk.
FaxDistribution	Specifies which Fax resource will be receiving the fax for a particular incoming call.
ISDNDistribution	Specifies which ISDN channel will be used for an outgoing call.

Management Parameter Definitions

Brief descriptions of the new Management parameters are listed alphabetically on the following pages.

AutoReceive

DESCRIPTION	Specifies whether inbound fax call should be automatically received if the GFXSHUTDOWN state values are set to 0 or 2. If the AutoReceive parameter is set to “yes”, inbound faxes are automatically received even when there is no application running of the channel to receive the fax. Inbound faxes received in this mode are stored as a queue record in the queue file. If the AutoReceive parameter is set to “no”, inbound faxes received when there are no applications running on the channel to receive them fail. The sender receives an error message indicating that the fax was not received.
SYNTAX	“yes” or “no”
DEFAULT	“no”
DATA TYPE	REG_SZ
REFERENCE	None

PriLayerEnable

DESCRIPTION	Specifies whether Transparent PRI support is enabled. If the parameter is set to “yes”, the Transparent PRI Support is enabled. If the parameter is set to “no”, the Transparent PRI Support is disabled.
SYNTAX	“yes” or “no”
DEFAULT	“no”
DATA TYPE	REG_SZ
REFERENCE	None

NumberOfTrunks

DESCRIPTION	Indicates the number of PRI trunks in the chassis. This parameter setting is ignored if the PriLayerEnable parameter is set to “no”.
SYNTAX	Numeric
DEFAULT	1
DATA TYPE	REG_DWORD
REFERENCE	None

ChannelsPerTrunk

DESCRIPTION	Indicates the number of ISDN channels per PRI trunk. This parameter setting is ignored if the PriLayerEnable parameter is set to “no”.
SYNTAX	Numeric
DEFAULT	30
DATA TYPE	REG_DWORD
REFERENCE	None

FaxDistribution

DESCRIPTION	Determines the selection of which Fax resource will be receiving the fax on a specific incoming call. The setting values include: 0 First free Fax resource starting with channel 1 1 Circular Fax resource assignment 2 First free Fax resource starting with the last channel 3 Same Fax channel as ISDN channel This parameter setting is ignored if the PriLayerEnable parameter is set to “no”.
SYNTAX	Numeric

FaxDistribution	
DEFAULT	0
DATA TYPE	REG_DWORD
REFERENCE	None

ISDNDistribution	
DESCRIPTION	Determines the selection of which ISDN channel will be used on a specific outgoing call. The setting values include: 0 First free ISDN channel counting from 0 1 Circular ISDN channel assignment 2 First free ISDN channel starting with the last channel 3 Same ISDN channel than Fax channel 4 TBD This parameter setting is ignored if the PriLayerEnable parameter is set to “no”.
SYNTAX	Numeric
DEFAULT	1
DATA TYPE	REG_DWORD
REFERENCE	None

ISDN Parameters

These parameters control the ISDN features. These parameter features are ignored if the PriLayerEnable parameter is set to “no”.

Table 11. ISDN Parameters

Parameter	Description
BC_xfer_cap	ISDN Make Call Block parameter.
Layer1_protocol	ISDN Make Call Block parameter.
BC_xfer_rate	ISDN Make Call Block parameter.
BC_xfer_mode	ISDN Make Call Block parameter.
origination_number_type	ISDN Make Call Block parameter.
origination_number_plan	ISDN Make Call Block parameter.
destination_number_type	ISDN Make Call Block parameter.
destination_number_plan	ISDN Make Call Block parameter.
origination_subnumber_type	ISDN Make Call Block parameter.
origination_subnumber_plan	ISDN Make Call Block parameter.
origination_phone_number	ISDN Make Call Block parameter.
destination_phone_number	ISDN Make Call Block parameter.
destination_subphone_number	ISDN Make Call Block parameter.
origination_subphone_number	ISDN Make Call Block parameter.
AcceptCallState	Determines whether to use ISDN Accept Call State or not.
PRI_Overlap_P1	Defines minimum number of digits needed before the board accepts incoming call in Overlap Receive mode.
PRI_Overlap_P2	Defines maximum wait time between ISDN SETUP pack and the first ISDN INFO packet.
CheckInBearer	Specifies whether to check the Bearer Compatibility during inbound SETUP.

Parameter	Description
BC_xfer_cap	ISDN Make Call Block parameter.
CheckInSetupFrame	Specifies whether to check the incoming SETUP packet.

ISDN Parameter Definitions

Brief descriptions of the new ISDN parameters are listed alphabetically on the following pages.

BC_xfer_cap

DESCRIPTION	ISDN Make Call Block parameter.
SYNTAX	Mnemonic value defined in the Dialogic DNA 3 online help.
DEFAULT	BEAR_CAP_SPEECH
DATA TYPE	REG_DWORD
REFERENCE	Refer to the Dialogic DNA3 Online Help for more information about the parameter settings.

Layer1_protocol

DESCRIPTION	ISDN Make Call Block parameter.
SYNTAX	Mnemonic value defined in the Dialogic DNA 3 online help.
DEFAULT	ISDN_UILI_G711ALAW
DATA TYPE	REG_DWORD
REFERENCE	Refer to the Dialogic DNA3 Online Help for more information about the parameter settings.

BC_xfer_rate

DESCRIPTION	ISDN Make Call Block parameter.
SYNTAX	Mnemonic value defined in the Dialogic DNA 3 online help.
DEFAULT	BEAR_RATE_64KBPS
DATA TYPE	REG_DWORD
REFERENCE	Refer to the Dialogic DNA3 Online Help for more information about the parameter settings.

BC_xfer_mode

DESCRIPTION	ISDN Make Call Block parameter.
SYNTAX	Mnemonic value defined in the Dialogic DNA 3 online help.
DEFAULT	ISDN_ITM_CIRCUIT
DATA TYPE	REG_DWORD
REFERENCE	Refer to the Dialogic DNA3 Online Help for more information about the parameter settings.

originate_number_type

DESCRIPTION	ISDN Make Call Block parameter.
SYNTAX	Mnemonic value defined in the Dialogic DNA 3 online help.
DEFAULT	ISDN_NOTUSED
DATA TYPE	REG_DWORD
REFERENCE	Refer to the Dialogic DNA3 Online Help for more information about the parameter settings.

originate_number_plan

DESCRIPTION	ISDN Make Call Block parameter.
SYNTAX	Mnemonic value defined in the Dialogic DNA 3 online help.
DEFAULT	ISDN_NOTUSED
DATA TYPE	REG_DWORD
REFERENCE	Refer to the Dialogic DNA3 Online Help for more information about the parameter settings.

destination_number_type

DESCRIPTION	ISDN Make Call Block parameter.
SYNTAX	Mnemonic value defined in the Dialogic DNA 3 online help.
DEFAULT	EN_BLOCK_NUMBER
DATA TYPE	REG_DWORD
REFERENCE	Refer to the Dialogic DNA3 Online Help for more information about the parameter settings.

destination_number_plan

DESCRIPTION	ISDN Make Call Block parameter.
SYNTAX	Mnemonic value defined in the Dialogic DNA 3 online help.
DEFAULT	UNKNOWN_NUMBER_PLAN
DATA TYPE	REG_DWORD
REFERENCE	Refer to the Dialogic DNA3 Online Help for more information about the parameter settings.

destination_subnumber_type

DESCRIPTION	ISDN Make Call Block parameter.
SYNTAX	Mnemonic value defined in the Dialogic DNA 3 online help.
DEFAULT	ISDN_NOTUSED
DATA TYPE	REG_DWORD
REFERENCE	Refer to the Dialogic DNA3 Online Help for more information about the parameter settings.

origination_subnumber_type

DESCRIPTION	ISDN Make Call Block parameter.
SYNTAX	Mnemonic value defined in the Dialogic DNA 3 online help.
DEFAULT	ISDN_NOTUSED
DATA TYPE	REG_DWORD
REFERENCE	Refer to the Dialogic DNA3 Online Help for more information about the parameter settings.

origination_phone_number

DESCRIPTION	ISDN Make Call Block parameter.
SYNTAX	Mnemonic value defined in the Dialogic DNA 3 online help.
DEFAULT	ISDN_NOTUSED
DATA TYPE	REG_DWORD
REFERENCE	Refer to the Dialogic DNA3 Online Help for more information about the parameter settings.

destination_sub_phone_number

DESCRIPTION	ISDN Make Call Block parameter.
SYNTAX	Mnemonic value defined in the Dialogic DNA 3 online help.
DEFAULT	ISDN_NOTUSED
DATA TYPE	REG_SZ
REFERENCE	Refer to the Dialogic DNA3 Online Help for more information about the parameter settings.

origination_subphone_number

DESCRIPTION	ISDN Make Call Block parameter.
SYNTAX	Mnemonic value defined in the Dialogic DNA 3 online help.
DEFAULT	ISDN_NOTUSED
DATA TYPE	REG_SZ
REFERENCE	Refer to the Dialogic DNA3 Online Help for more information about the parameter settings.

AcceptCallState

DESCRIPTION	Specifies whether or not to use the ISDN Accept Call State.
SYNTAX	“yes” or “no”
DEFAULT	“yes”
DATA TYPE	REG_SZ
REFERENCE	None

PRI_Overlap_Digits

DESCRIPTION	Defines the minimum number of digits needed before the board will accept the incoming call in Overlap Receiving mode.
SYNTAX	
DEFAULT	0
DATA TYPE	REG_DWORD
REFERENCE	None

PRI_Overlap_T1

DESCRIPTION	Defines the maximum time period to pause (in milliseconds) between sending the ISDN SETUP packet and the first ISDN INFO packet. Setting this parameter disables the Overlap Receiving parameter setting.
SYNTAX	
DEFAULT	2000
DATA TYPE	REG_DWORD
REFERENCE	None

PRI_Overlap_T2

DESCRIPTION	Defines the maximum time period to pause (in milliseconds) between sending a second ISDN INFO packet which contains additional routing digits in Overlap Receiving.
SYNTAX	
DEFAULT	2000
DATA TYPE	REG_DWORD
REFERENCE	None

CheckInBearer

DESCRIPTION	Indicates whether to check for compatible bearer capabilities in the incoming setup. If “yes”, checks for compatible bearer capabilities. If “no”, does not check bearer capability during inbound SETUP. Any bearer capability is accepted.
SYNTAX	
DEFAULT	“yes”
DATA TYPE	REG_SZ
REFERENCE	None

CheckInSetupFrame

DESCRIPTION	Indicates whether to check the incoming setup packet. If “yes”, checks the incoming setup packet. If “no”, does not check the incoming setup packet. Any setup packet is accepted.
SYNTAX	
DEFAULT	“yes”
DATA TYPE	REG_SZ
REFERENCE	None

ErrorMapping Parameters

These parameters control the ErrorMapping features. These parameter features are ignored if the PriLayerEnable parameter is set to “no”.

Table 12. ErrorMapping Parameters

Parameter	Description
ReservedByDialogic	Fax channel incoming/outgoing collision error.
OfferedOnMakeCall	ISDN channel incoming/outgoing collision error.
MakeCallFail	PRI is not able to complete the outgoing call operation error.
NoPhoneInQrec	Phone number field not correctly entered by application error, or is missing.
NoDialogicFree	Can not find PRI channel error.
CallDisconnected	Undefined ISDN connection error.
ConnectAttemptFail	Call released before connect state is reached error.
WrongCRNAllocated	Serious ISDN error.
NoFaxResource	Rejected call due to lack of an available Fax resource ISDN error.
FaxNotReady	Rejected call due to system not being initialized or in shutdown mode ISDN error.
NormalCause	Default ISDN error sent when releasing a call.
WrongBearer	Rejected call due to incompatibility in Bearer Compatibility ISDN error.

ErrorMapping Parameter Definitions

Brief descriptions of the new ErrorMapping parameters are listed alphabetically on the following pages.

ReservedByDialogic

DESCRIPTION	Error code that is returned if the application tries to place an outgoing call on a Fax channel that is reserved by the Dispatcher to accept an incoming fax.
SYNTAX	Any numeric value.
DEFAULT	3072
DATA TYPE	REG_DWORD
REFERENCE	None

OfferedOnMakeCall

DESCRIPTION	Error code that is returned if the application tries to place an outgoing call on a ISDN channel that is reserved to accept an incoming call.
SYNTAX	Any numeric value.
DEFAULT	3072
DATA TYPE	REG_DWORD
REFERENCE	None

MakeCallFail

DESCRIPTION	Error code that is returned if the PRI is not able to complete the outgoing call.
SYNTAX	Any numeric value.
DEFAULT	3932
DATA TYPE	REG_DWORD
REFERENCE	None

NoPhoneInQrec

DESCRIPTION	Error code that is returned if the phone number field is not entered correctly by the application, or is missing.
SYNTAX	Any numeric value.
DEFAULT	3933
DATA TYPE	REG_DWORD
REFERENCE	None

NoDialogicFree

DESCRIPTION	Error code that is returned if a PRI channel can not be found.
SYNTAX	Any numeric value.
DEFAULT	3932
DATA TYPE	REG_DWORD
REFERENCE	None

CallDisconnected

DESCRIPTION	Error code that is returned when an undefined ISDN disconnection is encountered.
SYNTAX	Any numeric value.
DEFAULT	3936
DATA TYPE	REG_DWORD
REFERENCE	None

ConnectAttemptFail

DESCRIPTION	Error code that is returned when a call is released before reaching the connect state.
SYNTAX	Any numeric value.
DEFAULT	3935
DATA TYPE	REG_DWORD
REFERENCE	None

WrongCRNAllocated

DESCRIPTION	Error code that is returned when there is a serious ISDN issue. Causes the Dialogic channel to be restarted.
SYNTAX	Any numeric value.
DEFAULT	3933
DATA TYPE	REG_DWORD
REFERENCE	None

NoFaxResource

DESCRIPTION	ISDN error returned when a call is rejected because there was no available Fax resource to receive it.
SYNTAX	Any numeric value.
DEFAULT	USER_BUSY
DATA TYPE	REG_DWORD
REFERENCE	None

FaxNotReady

DESCRIPTION	ISDN error returned when a call is rejected because the system is not initialized, or is in shutdown mode.
SYNTAX	Any numeric value.
DEFAULT	NORMAL_CLEARING
DATA TYPE	REG_DWORD
REFERENCE	None

NormalCause

DESCRIPTION	The default ISDN error returned when releasing the call.
SYNTAX	Any numeric value.
DEFAULT	NORMAL_CLEARING
DATA TYPE	REG_DWORD
REFERENCE	None

WrongBearer

DESCRIPTION	ISDN error returned when rejecting the incoming call because of an incompatibility in Bearer Capabilities.
SYNTAX	Any numeric value.
DEFAULT	CAP_NOT_IMPLEMENTED
DATA TYPE	REG_DWORD
REFERENCE	None

Debug Parameters

These parameters control the Debug features.

Table 13. Debug Parameters

Parameter	Description
UseSRAM	Specifies whether or not to enable Debug logging to Shared Memory in the GDK System Service.
SRAMMask	Mask value that specifies type of information logged to Shared Memory.
LogFile	Specifies file name to store Shared Memory Debug log information.
LogFileMask	Mask value that specifies type of information written to the file named by the Log File parameter.
DebugToSRAM	Specifies whether to allow DEBUG command information to be logged into shared RAM.
UseGFAX$DL	Specifies whether or not to create a log file named %GFAX%\GFAX.$DL.
TraceTrunkNumber	Defines the trunk number used for the ISDN tracing.
TraceFileName	Filename where all ISDN trace information is logged.

Debug Parameter Definitions

Brief descriptions of the new Debug parameters are listed alphabetically on the following pages.

UseSRAM

DESCRIPTION	Specifies whether or not to enable Debug logging to Shared Memory in the GDK System Service. If “yes” Shared RAM is used. If “no” Shared RAM is not used. This parameter setting is ignored if the PRILayerEnable parameter is set to “no”. This parameter is global across the CP Fax product line.
SYNTAX	“yes” or “no”
DEFAULT	“yes” if PRILayerEnable parameter is set to “yes”, “no” if the PRILayerEnable parameter is set to “no”
DATA TYPE	REG_SZ
REFERENCE	None

SRAMMask

DESCRIPTION	Mask value that specifies type of information logged to Shared Memory. This parameter setting is ignored if the UseSRAM parameter is set to “no”. This parameter is global across the CP Fax product line.
SYNTAX	bit mask value
DEFAULT	0xFFFFFFFF
DATA TYPE	REG_DWORD
REFERENCE	None

LogFile

DESCRIPTION	Specifies file name to store Shared Memory Debug log information. This parameter setting is ignored if the UseSRAM parameter is set to “no”. This parameter is global across the CP Fax product line.
SYNTAX	full path and filename.
DEFAULT	“”
DATA TYPE	REG_SZ
REFERENCE	None

LogFileMask

DESCRIPTION	Mask value that specifies type of information to file named by the Log File parameter. This mask does not affect the other information logged in Shared Memory. This parameter setting is ignored if the UseSRAM parameter is set to “no”. This parameter is global across the CP Fax product line.
SYNTAX	bit mask value
DEFAULT	0xFFFFFFFF
DATA TYPE	REG_DWORD
REFERENCE	None

DebugToSRAM

DESCRIPTION	Specifies whether DEBUG command information is logged to Shared Memory. The functionality of this parameter depends on the DEBUG command being set. If “yes” allows Debug information to be logged into Shared RAM (dependent on the GDK DEBUG Mask.) If “no” firmware information is not logged into the Shared RAM. When the set to “no”, the UseSRAM parameter settings are ignored. This parameter setting is ignored if the UseSRAM parameter is set to “no”. This parameter is global across the CP Fax product line.
SYNTAX	“yes” or “no”
DEFAULT	“no”
DATA TYPE	REG_SZ
REFERENCE	Refer to the Enabling Debug section in Chapter 2 of this guide

UseGFAX$DL

DESCRIPTION	Specifies whether or not to log Debug information to a text file called %GFAX%\GFAX.$DL. The functionality of this parameter depends on the DEBUG command being set. If “yes” Debug information is logged in GFAX.$DL (dependent of the GDK Debug mask.) If “no” debug information is not logged in the GFAX.$DL. This parameter setting is ignored if the UseSRAM parameter is set to “no”. This parameter is global across the CP Fax product line.

UseGFAX$DL

SYNTAX	"yes" or "no"
DEFAULT	"no"
DATA TYPE	REG_SZ
REFERENCE	Refer to the Enabling Debug section in Chapter 2 of this guide

TraceTrunkNumber

DESCRIPTION	Defines the trunk number (board handle) used for the ISDN tracing. This parameter setting is ignored if the PRILayerEnable parameter is set to "no".
SYNTAX	numeric
DEFAULT	1
DATA TYPE	REG_DWORD
REFERENCE	See Dialogic DNA 3 online manuals for more information on ISDN trace debugging.

TraceFileName

DESCRIPTION	File where the ISDN trace (cc_StartTrace function) information will be logged. If the filename is a NULL string, is invalid or has an invalid path, no ISDN path is provided. This parameter setting is ignored if the PRILayerEnable parameter is set to “no”.
SYNTAX	full path and filename
DEFAULT	“”
DATA TYPE	REG_SZ
REFERENCE	See Dialogic DNA 3 online manuals for more information on ISDN trace logging.

4. Queue Record Programming

Queue File Database Component

This section discusses the Queue File database component of the GDK system software. This section discusses the following:

- Overview of the Queue File
- Handling the Queue File
- Record queuing and processing
- Buffering records
- Busy records
- Fax transaction programming
- Queue record data types
- Queue record fields

About the Queue File

The Queue File, GFAX.$QU, is the database that contains a detailed record of information on each incoming and outgoing transaction. It is made up of a header and seven linked lists of records, and is located in the subdirectory defined by the GFAX environment variable.

A queue record is a data structure in the Queue File that permits an application program to communicate with the firmware. The fields in a queue record provide information about the image to be sent and where and how to send it, or when and how an image was received. Some diagnostic information is also stored in the queue record.

Queue File Lists

The Queue File contains seven linked lists:

- Free List
- Pending List
- Received List
- Sent List
- Conversion List
- Control List
- Control Done List

Free List

The Free List collects deleted and unused records for future use and re-allocation. When a new transaction is inserted into the Queue File, the Free List is checked for available records. If a record is available, it is overwritten, and "linked" into the appropriate linked list. Whenever records from the Free List are used, the Queue File does not grow in size. If there are no records available in the Free List, a new record is appended to the Queue File, and linked to the appropriate list — expanding the size of the Queue File. When a record is deleted from a linked list, it is placed in the Free List, and the size of the Queue File remains the same.

Pending List

The Pending List stores records submitted to the Queue File for transmission. Records waiting for processing are sorted first by time, and then by the priority level.

Received List

The Received List stores all completed incoming transmissions. Records in the Received List have a status code that indicates whether or not the operation was successful.

Sent List

The Sent List is used to store completed outgoing transactions. Once a transmission record on the Pending List has finished processing, it is moved from the Pending List to the Sent List. Records in the Sent List have status codes indicating whether or not the operation was successful.

Conversion List

The Conversion List is intended to be used by applications as a holding area for those queue records needing additional processing before being submitted to the Pending List. This is a good place to put non-TIFF files needing to be converted to TIFF format. The application is responsible for checking this list for jobs and submitting the file to be converted to a third-party conversion utility.

Control and Control Done Lists

When the GDK System Service processes certain configuration commands, it generates records in the Control List for each fax channel. These records initialize the system, and are moved to the Control Done List when initialization is complete.

Queue File Pointers

The functions, gfqFindFirst() and gfqFindNext(), access the linked lists in the Queue File and allow you to traverse the Queue File lists and read information from the records. These functions are described in Chapter 5.

Using GFQRESET.EXE

The GFAX environment variable is always used to determine the path for the Queue File; the default filename is “GFAX.$QU.” If a Queue File does not exist, the program GFQRESET.EXE creates one.

In addition to creating a Queue File, the GFQRESET.EXE program performs the following:

- Clears all busy records
- Pre-allocates records to a Queue File

- Purges old records from the Control and Control Done lists
- Checks or repairs the Queue File link lists

This is the usage for GFQRESET:

```
GFQRESET -pn -qn -rn <queue name>
```

The parameters are listed in Table 14 , and are described in more detail in the following section.

Table 14. GFQRESET Parameters

Option	Function
-pn	The purge flag option; these are the values for “n”: 0 = No purging. (Default) 1 = Purge Control Done List. 2 = Purge Control List. 3 = Purge Control Done and Control Lists.
-qn	The Queue File check option; these are the values for “n”: 0 = No queue file check. (Default) 1 = Queue file is checked and not repaired. 2 = Queue file is checked and repaired.
-rn	The pre-allocation and prevent-collapse flag. The range for “n” is 0 to 32,767. 0 = Queue file may be collapsed.
<queue name>	The name of the Queue File to create or reset.

Pre-allocating Queue File Records

Space can be reserved in the Queue File by pre-allocating Queue File records with the -r option. Preallocated records are stored in the Free List of the Queue File, which holds “empty” records available for transaction data. Preallocation is used to allocate contiguous queue record storage to avoid file fragmentation.

NOTE: Each record requires 516 bytes.

If a Queue File does not exist, it is created and the specified number of records is inserted into the Free List. If a Queue File already exists, the queue records are counted and the program adds more records if needed. If the Free list of preallocated records is exhausted, the next record is appended to the Queue File.

Purging the Control List and Control Done List

Records in the Control List and Control Done List are not needed after the GDK system has been initialized. To purge these lists so the Queue File does not become too large, use this command:

```
GFQRESET -p3
```

NOTE: The GDK Service runs GFQRESET with a -p3 option automatically upon startup. GFQRESET does not need to be called manually, unless other options are desired. If desired, GFQRESET should be called before the GDK Service is started.

Checking and Repairing the Queue File

With the -q option (Table 14), the Queue File can be checked and repaired if any records have been damaged. If an error is returned by the -q option, backup or delete the Queue File and restart the system.

CAUTION

Using the -q2 option may not always work and should not be attempted with the Dispatcher service running.

Record Queuing and Processing

The initial channel state depends on the GFXSHUTDOWN command. With the GFXSHUTDOWN command parameter set to 3, a fax channel ignores queue records in the Pending List and ignores incoming calls. The application must submit reception/transmission requests using the GRT programming model.

With the GFXSHUTDOWN command parameter set to 0, a fax channel may become ready for reception/transmission at any time, depending on the number of channels in the system and how busy they are at any given moment.

When a fax channel becomes idle, the firmware requests work through the Dispatcher. The Dispatcher retrieves the queue file records from the queue file. If a queue record exists, it is given to the target channel, and the queue record on disk is updated with the new status. If no record is available, a message indicating that nothing is pending is sent to the fax channel.

The fax channel performs the task specified by the operation field of the record. The firmware communicates with the Dispatcher about the files or other information it may need, and the Dispatcher provides details to the firmware. The firmware then transmits the fax and provides an updated record to the Dispatcher. The Dispatcher posts the updated record to the Queue File, which stores a history of the fax transmission — received, sent, failed, or awaiting retransmission.

Buffering Records

The current implementation utilizes buffering of the queue records. Recommended buffers setting is two times the number of fax channels. Buffers are used for storing queue records in RAM before being requested by an idle fax channel. Buffering queue records reduces the number of disk accesses to the Queue File.

Busy Records

When the QUEUET timer expires, the Dispatcher starts at the beginning of the Pending List. The Dispatcher then loads and marks “BUSY” as many ready-to-go queue records as it has fax channels available.

CAUTION

If a “BUSY” Pending List record is deleted, results may be unpredictable.

Although the original record is undisturbed in the Queue File, a record marked "BUSY" indicates that a copy of the original record is in the record buffer of the Dispatcher or is held by the firmware.

Fax Transaction Programming

This section discusses fax transaction programming with GDK queue records. A *queue record* is a C data structure that can be manipulated by the application program. The fields in a queue record provide complete information about the image to be sent or received. These fields also contain information on where and how to send the image or when and how it was received. In addition, many of the fields also provide diagnostic information.

Each queue record contains information on a single telephone call. The records can also be used to install default communication settings for answer operations, such as Answer and Send or Answer and Send/Receive. In this case, the original record is then used as a template to post new records for incoming calls.

Queue Record Data Types

The GFQ.H queue record header file contains preprocessor directives, data-type definitions, and error codes for the GDK system.

Table 15. GFQ.H Data Types

Mnemonic	Data Type	Length	Purpose
GFQOFFSET	unsigned long	4 bytes	Offset from beginning of file.
GFQLONG	unsigned long	4 bytes	
GFQTIME	unsigned long	4 bytes	Date/time stamps in seconds.
GFQBYTE	unsigned char	1 byte	Characters.
GFQINT	unsigned short	2 bytes	
GFQFILENAME_SIZE	N/A	64 + 2 bytes	For filenames.
GFQCSID_SIZE	N/A	20 + 2 bytes	For phone numbers.

Mnemonic	Data Type	Length	Purpose
GFQUSER_FIELD_SIZE	N/A	32 + 2 bytes	For (network) addresses.

N/A = not applicable

Queue Record Fields

This section summarizes all of the queue record fields. It also provides an overview table and an alphabetic listing of the queue record fields with a description of their parameters.

Queue Record Field Descriptions

Table 16 provides a synopsis of the queue record fields.

NOTES: **1.** Only a few fields need to be changed from the default values set by gfqClearRec( ).

2. All queue record fields which contain filenames must conform to the MS-DOS naming convention (eight characters separated by a dot, and then followed by a three-character extension).

Table 16. Summary of Fields in the Queue Record

Field	Size in Bytes	Data Types	Use
cd_timeout	2	GFQINT	Optional
completed_retries	2	GFQINT	Managed by GDK[1]
completed_time	4	GFQTIME	Managed by GDK[1]
control	2	GFQINT	Optional
csid	$20 + 2^3$	GFQBYTE	Optional
curr	4	GFQOFFSET	Managed by the libraries[2]
duration	2	GFQINT	Managed by GDK[1]
fn_cover	$64 + 2^3$	GFQBYTE	Optional
fn_received	$64 + 2^3$	GFQBYTE	Managed by GDK[1]
fn_send	$64 + 2^3$	GFQBYTE	Required (for sending)

Field	Size in Bytes	Data Types	Use
header	20	GFQBYTE	Optional
items_received	2	GFQINT	Managed by GDK[1]
items_sent	2	GFQINT	Managed by GDK[1]
line_noise	2	GFQINT	Managed by GDK[1]
list_type	2	GFQINT	Managed by GDK[1]
message_speed	2	GFQINT	Managed by GDK[1]
modem_id	32+2[3]	GFQBYTE	Optional
next	4	GFQOFFSET	Managed by the libraries[2]
notify	2	GFQINT	Managed by GDK[1]
nsf_field	32+2[3]	GFQBYTE	Optional
nsf_length	2	GFQINT	Optional
number_calls	2	GFQINT	Optional
operation	2	GFQINT	Required
phone_no	20+2[3]	GFQBYTE	Required (for sending), with call control
prev	4	GFQOFFSET	Managed by the libraries[2]
priority_level	2	GFQINT	Optional
protocol	2	GFQINT	Optional
rate	2	GFQINT	Optional
received_csid	20+2[3]	GFQBYTE	Managed by GDK[1]
received_filetype	2	GFQINT	Managed by GDK[1]
received_nsf	32+2[3]	GFQBYTE	Managed by GDK[1]
received_nsf_length	2	GFQINT	Managed by GDK[1]
record_control	2	GFQINT	Managed by GDK[1]
retry_counter	2	GFQINT	Optional
retry_delay	2	GFQINT	Optional
retry_end_time	4	GFQTIME	Optional
retry_strategy	2	GFQINT	Optional

Field	Size in Bytes	Data Types	Use
security	4	GFQLONG	Optional
signal_quality	2	GFQINT	Managed by GDK[1]
signal_strength	2	GFQINT	Managed by GDK[1]
source_type	2	GFQINT	Optional
status	2	GFQINT	Managed by GDK[1]
submission_retries	2	GFQINT	Managed by GDK[1]
submission_time	4	GFQTIME	Managed by GDK[1]
time	4	GFQTIME	Optional
transmit_control	2	GFQINT	Optional
user_id	32+2[3]	GFQBYTE	Optional

1 This field is filled in by the firmware after it processes the job. However, it may also be read from or written to by the GDK functions.

2 Both the firmware and the GDK functions manage the values in these fields, and you should not write directly to them.

3 Two bytes are used for terminating null characters for the string.

Alphabetical Listing of Queue Record Fields

The field descriptions are presented in the format shown in Table 17.

Table 17. Queue Record Field Description Formats

Function Name	
DATA TYPE	The GDK data type description.
DESCRIPTION	An overview of the purpose of the field.
VALUE	Indicates the values required by the field.
USAGE	Provides an example of how to use this field in the GDK.
SEE ALSO	Refers to other queue record fields that contain additional or related information.

cd_timeout

DATA TYPE	GFQINT
DESCRIPTION	This field specifies the number of seconds to wait after dialing for answer-tone carrier detect. It cannot have a value of zero; the default is 30 seconds. On overseas calls, the connect time may be 60 to 90 seconds. Local calls might connect in 10 seconds.
VALUE	Positive integers to 32767.
USAGE	qrec->cd_timeout = 45;

completed_retries

DATA TYPE	GFQINT
DESCRIPTION	This field contains the number of completed retries and is managed by GDK firmware. For example, if the submission_retries is 3 and the retry_counter is 2, the completed_retries value is 1.
VALUE	Positive integers to 32767.
USAGE	attempts = qrec->completed_retries;

completed_time

DATA TYPE	GFQTIME
DESCRIPTION	This field tells when a record was completed and posted to a list. For example, after a record in the Pending List has been processed, it is posted to the Sent List and the completed_time is recorded in seconds. It is managed by GDK firmware.
VALUE	Positive long integer.
USAGE	ptr = ctime(&(qrec->completed_time));

control

DATA TYPE	GFQINT
DESCRIPTION	This field is used for Answer & Send, Answer & Receive, and Answer Default records only. A record waits for the phone to ring for the specified amount of time before it expires with a status code of 3033. The record is posted to the queue or application notification pipe after the specified time. The value is given in seconds; zero can be used. The default is 60.
VALUE	Positive integers to 32767.
USAGE	qrec->control = 20;

csid

DATA TYPE	GFQBYTE[GFQCSID_SIZE]
DESCRIPTION	This field is transmitted as the Customer Subscriber Identification (CSID) number during T.30-protocol handshaking. Usually, it contains the telephone number of the station or installation. It holds a maximum of 20 characters, or it can be empty. If the CSID field has spaces, it remains blank. Writing more than 20 characters to this field may produce unexpected results. If the CSID field is null and does not have spaces, the CSID from the user's configuration file corresponding to the fax channel processing the record is placed into the field. PTT administrations in some countries may restrict characters in this field to the numbers 0 through 9 and the plus (+).
VALUE	Up to 20 alphanumeric characters.
USAGE	strcpy (qrec->csid,"408-744-1549");

curr

DATA TYPE	GFQOFFSET
DESCRIPTION	This field contains a pointer to the current queue record in a linked list, and is managed by GDK firmware. When a queue record is inserted into the Queue File, the current record pointer is "assigned."
VALUE	Positive integer.
USAGE	This field is managed by GDK software and should not be modified by user programs.

duration

DATA TYPE	GFQINT
DESCRIPTION	This field can be used to determine the length of time it took to complete a phone call. It records the number of seconds of phone-connect time used for transmission or reception. In a Dial-and-Send-type operation, the timer starts at the end of the answer tone. This field is managed by the GDK communication programs.
VALUE	Positive integers to 32767.
USAGE	phone_time = qrec->duration;

fn_cover

DATA TYPE	GFQBYTE[GFQFILENAME_SIZE]
DESCRIPTION	This 64-byte field contains the name of the file comprising the cover page, if desired. It must be used in conjunction with the field (transmit_control |= GFQUSE_COVERSHEET).
VALUE	A character array of type GFQFILENAME_SIZE; always a full path to the cover-page file.
USAGE	strcpy (qrec->fn_cover,"c:\\fax\\cover.tif");
SEE ALSO	transmit_control

fn_received

DATA TYPE	GFQBYTE[GFQFILENAME_SIZE]
DESCRIPTION	This field contains the filename of a received fax file. The filename can be up to 64 bytes in length. The communication program initializes this field after the fax is received. A default filename is given to received transmissions — "f001p001.tif," in which the numerals increment with the transmission and page numbers. The first character indicates the fax channel that received the file. The first group of digits is a "call number," which provides a unique number for the fax. The character "p" refers to page, and the second group of digits indicates the page number of the fax. The default filename can be changed when a queue record is submitted with an Answer-operation parameter and by filling in this field. It may also be changed by using the GFXRECVPATH command. However, the first character always represents the fax channel that received the fax. The type of file it contains is indicated in the received_filetype field. Usually, the received_filetype is GFQSINGLE_DOC; however, for file transfer, the fn_received field generates a received_filetype of GFQLIST_OF_DOCS. If file transfer is used, the received filename is a list of files in the format xxxxXFER. FLS. This list contains the name of the file that was received, which is the same as the original sent filename (as long as the DOS xxxxxxxx.yyy format is observed and the filename does not already exist on the target drive). The "f001p001" filenaming style is used if the sent filename already exists; the original filename is included in parentheses next to the filename that was written in the list.

fn_received (cont.)

	For example, A001XFER.FLS could contain: TEST.TXT A001P001.TIF (TEST.PCX)
VALUE	A character array of type [GFQFILENAME_SIZE];.
USAGE	strcpy (qrec->fn_received, "c:\\fax\\f001p001.tif");
SEE ALSO	received_filetype

fn_send

DATA TYPE	GFQBYTE[GFQFILENAME_SIZE]
DESCRIPTION	This field, which can be up to 64 bytes in length, contains the name of a file to be sent. The type of file to be sent should be put in the source_type field. If it is a single file or a sequence of pages in the GDK filename format F001P001.TIF, the source_type is GFQSINGLE_DOC. If it is a list of files, the source_type is GFQLIST_OF_DOCS. A list of files must be an ASCII file, with every line containing a path to a file to send. Each line must be separated by a <CR><LF>.
VALUE	A character array of type [GFQFILENAME_SIZE];.
USAGE	strcpy (qrec->fn_send, "c:\\fax\\f001p001.tif");
SEE ALSO	source_type

header

DATA TYPE	GFQBYTE[GFQCSID_SIZE]
DESCRIPTION	This field contains the text that is included in a header, which is printed at the top of each sent fax. It is limited to 20 alphanumeric characters, but also can be empty. This field must be used in conjunction with (transmit_control |= GFQUSE_HEADER). The following is the default format of the header line on the printed page: date & time calling CSID -> called CSID text page # The information in this field fills the "text" field in the header file.
VALUE	A character array of type [GFQCSID_SIZE].
USAGE	strcpy (qrec->header,"Fax from GDK");
SEE ALSO	transmit_control

items_received

DATA TYPE	GFQINT
DESCRIPTION	This field contains the number of pages or files correctly received. If nothing is received because of an error, this field contains zero.
VALUE	Positive integers to 32767.
USAGE	number_pages = qrec->items_received;

items_sent

DATA TYPE	GFQINT
DESCRIPTION	This field contains the number of pages or files actually sent during the transmission, and it is managed by GDK firmware. If nothing is sent because of an error, this field contains zero.
VALUE	Positive integers to 32767.
USAGE	number_pages = qrec->items_sent;

line_noise

DATA TYPE	GFQINT
DESCRIPTION	The line noise, which is measured by the modem during quiet periods of the handshaking process, is recorded here. It is filled by GDK firmware when a message is received. This field is useful for determining if a failure in transmission is due to line problems.
VALUE	See Table 18.
USAGE	noise = qrec->line_noise

Table 18. Values Reported in line_noise

Value	dBm	Meaning
18	-65	Extremely quiet
50	-60	Very quiet
135	-55	Quiet
461	-50	Acceptable
1390	-45	Noisy
2300	-40	Unusable

list_type

DATA TYPE	GFQINT
DESCRIPTION	This field indicates the list to which a queue record belongs; it is managed by GDK firmware.
VALUE	See Table 19.
USAGE	if (qrec->list_type == GFQPEND_LIST) printf ("Pending Log");

Table 19. list_types in the Queue Record

Mnemonic	Meaning
GFQPEND_LIST	Pending List
GFQRECV_LIST	Received List
GFQSENT_LIST	Sent List
GFQCONV_LIST	Conversion List
GFQCTRL_LIST	Control List
GFQCPST_LIST	Control Done List
GFQFREE_LIST	Free (deleted) Records
GFQALL_LISTS	All Lists

message_speed

DATA TYPE	GFQINT
DESCRIPTION	This field records the transmission rate in bits per second (BPS) during transmission or reception. It is managed by GDK firmware.
VALUE	14400, 12000, 9600, 7200, 4800, and 2400 bps or GFQMAX_RATE for maximum default rate.
USAGE	speed = qrec->message_speed;

modem_id

DATA TYPE	GFQBYTE[GFQUSER_FIELD_SIZE]
DESCRIPTION	This 32-byte field is used for a multiple-fax channel chassis. It holds a string in the form “GFAXx.yy,” where “x” represents the chassis number and “yy” is the channel number. (For more than nine channels, numbers in the form of “01,” and so on, should be used.) This string should match the one used with the channel command in the ChannelID key in the registry file. When a queue record is submitted, the modem_id indicates the channel to which the task should be assigned. If this field is empty, the task goes to the first available channel.
VALUE	A character array of size [GFQUSER_FIELD_SIZE].
USAGE	strcpy (qrec->modem_id,"GFAX1.01"); strcpy (qrec->modem_id,"GFAX1.10");

next

DATA TYPE	GFQOFFSET
DESCRIPTION	This field contains a pointer to the next queue record in a linked list and is managed by GDK firmware. The last record in a linked list contains NULL.
VALUE	Positive integer.
USAGE	This field is managed by GDK software and should not be modified by user programs.

notify

DATA TYPE	GFQINT
DESCRIPTION	This field is evaluated by gfqClearReq( ); otherwise, it is unused. If your application writes to this field, it is not evaluated.
VALUE	None.
USAGE	This field is managed by the GDK software and should not be modified by user programs.

nsf_field

DATA TYPE	GFQBYTE[GFQUSER_FIELD_SIZE]
DESCRIPTION	This is the non-standard-facilities (NSF) field to be transmitted during T.30 handshaking. It can hold up to 32 bytes and is reserved for facsimile machines that support NSF. The ITU specifies that the first octet of the NSF/NSS/NSC frame must contain an ITU country code. The next two octets contain a provider code. The GDK provider code is 0x00,0x64.
VALUE	A character array of size [GFQUSER_FIELD_SIZE].
USAGE	qrec->nsf_field[0] = 0xAE;
SEE ALSO	nsf_length

nsf_length

DATA TYPE	GFQINT
DESCRIPTION	This field contains the number of bytes to be transmitted in the non-standard-facilities (NSF) field. It is sent to facsimile machines that support NSF.
VALUE	Positive integer of size 0 to GFQUSER_FIELD_SIZE.
USAGE	qrec->nsf_length = 20;
SEE ALSO	nsf_field

number_calls

DATA TYPE	GFQINT
DESCRIPTION	This field specifies the number of dial attempts to be made for each "retry." For example, if number_calls is equal to 3 for a queue record with 3 retries, the board will attempt to dial up to 9 times. If this field is not specified when a queue record is submitted, the default is 1.
VALUE	Positive integers to 32767.
USAGE	qrec->number_calls = 3;

operation

DATA TYPE	GFQINT
DESCRIPTION	When a record is submitted, this field must specify the operation to be performed. The operation tells the on-board software how to process a queue record.
VALUE	See Table 20.
USAGE	qrec->operation = GFQDIAL_SEND;

Table 20. Queue-Record Operations

Operation	Transaction	Send DIS	Wait for DIS
DIAL	Send		GFQDIAL_SEND
	Receive		GFQDIAL_RECEIVE
	Both		GFQDIAL_SEND_ RECEIVE
ANSWER	Send	GFQANSWER_SEND	
	Receive	GFQANSWER_RECEIVE	
	Both	GFQANSWER_RECEIVE _SEND	
IMMEDIATE	Either	GFQANSWER_ IMMEDIATELY	

phone_no

DATA TYPE	GFQBYTE[GFQCSID_SIZE]
DESCRIPTION	This field contains the telephone number to be dialed. It is limited to 20 characters (Table 20); no spaces are permitted. Characters such as "-", "(", and ")" count toward the 20-character limit, but are not evaluated. For additional dialing features, see the Routing section in Chapter 2. For dial strings with more than 20 digits, put the dial string into an ASCII file. The first character of the phone_no field is checked for the character "@." If an "@" is found, the remainder of the field (up to the first white space) is used as the name of the file from which the phone digits are to be read. The file to be used must be a fully-qualified path name. If the file specified cannot be found, or if it appears to be a non-ASCII file, the call is terminated with no retries. If the characters "P" or "T" are used, the modem changes to pulse or tone, respectively.
VALUE	A character array of size [GFQCSID_SIZE]. See Table 21.
USAGE	strcpy (qrec->phone_no,"408-744-1549").

Table 21. Characters in phone_no Field

Character(s)	Meaning
0 through 9	Phone numbers and access numbers
A through D	Phone numbers and access numbers
, (comma)	1.5-second pause
; (semicolon)	15-second wait for second dial tone
P	Pulse dialing
T	Tone dialing (default)
!	Flash-hook signal
#	Notifies international operator that the dialing sequence is finished
*	PBX-specific information

prev

DATA TYPE	GFQOFFSET
DESCRIPTION	This field contains a pointer to the previous queue record in a linked list. It is managed by GDK firmware. The first record contains NULL.
VALUE	Any positive integer.
USAGE	This field is managed by the GDK software and should not be modified by user programs.

priority_level

DATA TYPE	GFQINT
DESCRIPTION	The records in the Queue File are sorted first by date and time stamp (specified in the time data field), then by priority, then processed sequentially. Although records may be inserted non-sequentially, those submitted with the same date and time stamp are processed in the order of submission, unless priority is specified. Records with the priority date and time specified are processed first. Zero is the lowest priority; 32767 is the highest. The default is zero.
VALUE	Positive integers to 32767.
USAGE	qrec->priority_level = 99;

protocol

DATA TYPE	GFQINT
DESCRIPTION	Two modes of communication are available on CP Fax Series boards: sending an image file to a fax machine (GFQT30_PROTOCOL) and sending a data file. GFQT30_PROTOCOL is the default; it should be changed only if file-transfer mode is desired.The method used for data transfer is GFQBFTS BFT T.434 protocol. To use BFT, additional configuration commands must be placed in the registry. See the GFXFAXCONTROL 1020.
VALUE	GFQT30_PROTOCOL, GFQFILE_TRANSFER,

protocol

DATA TYPE	GFQINT GFQBFT
USAGE	qrec->protocol = GFQBFT;

rate

DATA TYPE	GFQINT
DESCRIPTION	This field is used to initialize the rate of transmission in bits per second (bps). If it is set to GFQMAX_RATE, the message is to be sent at the highest rate possible. Change the value only if a slower rate is required; for example, when a phone line is known to have poor quality.
VALUE	GFQMAX_RATE, 14400, 12000, 9600, 7200, 4800, and 2400 bps.
USAGE	qrec->rate = 4800;

received_csid

DATA TYPE	GFQBYTE[GFQCSID_SIZE]
DESCRIPTION	This field records the Customer Subscriber Identification (CSID) number of the remote machine. It is up to 20 characters long, can contain any alphanumeric character, and is managed by GDK firmware.
VALUE	A character array of size [GFQCSID_SIZE].
USAGE	strcpy (station_id,qrec->received_csid);

received_filetype

DATA TYPE	GFQINT
DESCRIPTION	This field contains a description of the file named in fn_received. When a transmission is received, it is used to indicate whether the operation involves a single fax, a list of files, or a file transfer. The field is managed by GDK firmware.
VALUE	GFQSINGLE_DOC or GFQLIST_OF_DOCS.
USAGE	if (qrec->received_filetype == GFQLIST_OF_DOCS) printf ("List of files was received.\n")
SEE ALSO	fn_received

received_nsf

DATA TYPE	GFQBYTE[GFQUSER_FIELD_SIZE]
DESCRIPTION	This 32-byte field contains the value of the non-standard-facilities (NSF) field received from the sending facsimile machine. It is managed by GDK firmware.
VALUE	A character array of type [GFQUSER_FIELD_SIZE].
USAGE	for (i = 0; i < qrec->received_nsf_length; i++) printf ("%02x", qrec->received_nsf);
SEE ALSO	received_nsf_length

received_nsf_length

DATA TYPE	GFQINT
DESCRIPTION	This field contains the length of the non-standard-facilities (NSF) field that was received. It is reserved for communication with facsimile machines that support NSF. It is managed by GDK firmware.
VALUE	Integers from 0 to GFQUSER_FIELD_SIZE.
USAGE	This field is managed by the GDK software and should not be modified by user programs.
SEE ALSO	received_nsf

record_control

DATA TYPE	GFQINT
DESCRIPTION	This is a 16-bit field used for management of the queue record. The bits used by GDK functions are 0, 1, 2, 4, 5, 6, and 7. Bit 0 is "0" (zero) when the record is not busy and "1" when the record is busy. Bit 2 is "0" (zero) when the record is not "off the host" and "1" when the record is "off the host." When a record is submitted to the Pending List for transmission, its record_control value is "0" (zero). When it is moved by the Dispatcher into a buffer, it is "1." When the Dispatcher sends it to the on-board software for transmission, it changes to "5." When the record is posted and the transmission is complete, the record_control value is "0" (zero). During the course of a fax transmission, the record_control field goes through a series of steps. At each step, the values in this field change. See Table 22.
VALUE	See Table 23.
USAGE	This field is managed by the GDK software and should not be modified by user programs.
SEE ALSO	"Busy Records" section.

Table 22. Transmission Steps and record_control Field Values

Step	Values
A record is submitted to the Pending List.	xxxx x0x0
The Dispatcher moves the record into a buffer.	xxxx x0x1
The Dispatcher sends the record to the fax channel.	xxxx x1x1
The record is posted.	xxxx x0x0

Table 23. Values and Flags of the record_control Field

Bit	Bit Value of 0	Bit Value of 1	Flag Name
b0	Queue record not busy	Queue record busy	GFQRECORD_BUSY
b1	Template	Transaction	GFQTEMPLATE_RECORD
b2	On host	Off host	GFQRECORD_ON_HOST
b3	(Reserved)		
b4	No post	Post	GFQPOST_RECORD
b5	Resubmit off	Resubmit on	GFQRESUBMIT_ON
b6	Not viewed/printed	Viewed/printed	GFQRECORD_VIEWED
b7	Not routed	Routed	GFQRECORD_ROUTED
...			
b15	(Reserved)		

retry_counter

DATA TYPE	GFQINT
DESCRIPTION	This field counts the call-processing submissions, even though it is called a “retry” field. It specifies the total number of calls to be attempted, and not the number of “retries.” However, if the value of the field is set to zero, a file gets one transmission attempt by default. Because every record gets at least one transmission attempt, this field counts that first attempt as a “retry.” A value does not have to be assigned to retry_counter, unless a number other than the default is desired. If the -1 option is selected, retrying continues indefinitely, or for a specified period of time. However, the maximum value of the retry_counter may be restricted by PTT requirements in certain countries. This is only used in the Queue Programming model.
VALUE	-1, positive integers to 32767.
USAGE	qrec->retry_counter = 3;

retry_delay

DATA TYPE	GFQINT
DESCRIPTION	This field specifies the number of minutes that must elapse before another retry is attempted. The default is 30 minutes. This field does not have to be initialized unless this value is to be changed. PTT requirements in some countries may specify a minimum retry_delay time. This is only used in the Queue Programming model.
VALUE	Positive integers to 32767.
USAGE	qrec->retry_delay = 30;

retry_end_time

DATA TYPE	GFQTIME
DESCRIPTION	When a queue record is submitted for polling operations, it must have a start and stop time. This field indicates the stop time in seconds. If the record has not been processed successfully by its stop time, processing is discontinued, and the record is posted to the appropriate completion list. In that event, the retry_counter is set to -1. This is only used in the Queue Programming model. To specify a wait time for answer-operations in the GRT-based programming model, use the “control” field.
VALUE	Positive integers to 32767.
USAGE	qrec->retry_end_time = qrec->time + 3600;
SEE ALSO	retry_counter

retry_strategy

DATA TYPE	GFQINT
DESCRIPTION	If a record fails, the Queue Manager uses the type of retry strategy specified in this field The only retry strategy implemented at this time is specified with the parameter GFQFULL_RETRY.
VALUE	GFQFULL_RETRY.
USAGE	qrec->retry_strategy = GFQFULL_RETRY;

security

DATA TYPE	GFQLONG
DESCRIPTION	This field is used for checking the answering Customer Subscriber Identification (CSID) against the telephone number that was dialed. It contains the number of digits to be checked. One or all digits dialed, except for dashes and commas, can be checked against the CSID supplied by the called party to verify that the correct number was reached. A match of "n" numbers is performed; if there is no match, your system disconnects. If logging is enabled at the appropriate level, the message "Security check failed" is returned. A value of zero indicates no security.
VALUE	None.
USAGE	qrec->security = 3;

signal_quality

DATA TYPE	GFQINT
DESCRIPTION	This field contains the signal quality, which is measured by the modem during handshaking procedures. This field is useful for determining if a failure in transmission is due to line problems.
VALUE	See Table 24.
USAGE	This field is managed by the GDK software and should not

signal_quality

DATA TYPE GFQINT
be modified by user programs.

Table 24. Values of the signal_quality Field

Speed	Mode	High	Average	Marginal
14400	V.17	0	768	1024
12000	V.17	0	1792	2560
9600	V.17	0	4608	5632
7200	V.17	0	7168	10240
9600	V.29	0	960	2048
7200	V.29	0	2112	4352
4800	V.27	0	1600	3328
2400	V.27	0	9408	18944

signal_strength

DATA TYPE	GFQINT
DESCRIPTION	This field holds the signal strength, which is measured by the modem during handshaking. It is useful for determining if a failure in transmission is due to line problems.
VALUE	See Table 25.
USAGE	This field is managed by GDK software and should not be modified by user programs.

Table 25. Values of the signal_strength Field

Value	dBm	Strength
6750	-40	Marginal
10000	-35	Weak
13250	-30	Acceptable
16250	-25	Good
19750	-20	Strong
23000	-15	Very strong

source_type

DATA TYPE	GFQINT
DESCRIPTION	When a file to be sent is submitted to the Queue File, the user must specify whether the operation involves a single fax or a list of documents. This field contains a description of the file named in fn_send. It must be filled if a file is to be sent.
VALUE	GFQSINGLE_DOC or GFQLIST_OF_DOCS.
USAGE	qrec->source_type = GFQLIST_OF_DOCS;
SEE ALSO	fn_send

status

DATA TYPE	GFQINT
DESCRIPTION	This field contains a code indicating the success or failure of the transaction. A list of error and status codes is given in the *Error and Status Codes Manual*. Although the function libraries write the status field, codes can be written to it easily for your application.
VALUE	Positive integers to 32767.
USAGE	result = qrec->status;
SEE ALSO	*Error and Status Codes Manual*

submission_retries

DATA TYPE	GFQINT
DESCRIPTION	This field is used to manipulate retries, and contains the initial value of the retry_counter field. It records the number of retry attempts to be made when a transmission fails. This field is managed by GDK firmware.
VALUE	Positive integers to 32767.
USAGE	This field is managed by the GDK software and should not be modified by user programs.

submission_time

DATA TYPE	GFQTIME
DESCRIPTION	This field is used to manipulate submission times. It records the time at which a queue record was first submitted to the Queue File. The value in the field is the time in seconds from January 1, 1970. This field is managed by GDK firmware.
VALUE	A date/time stamp comprised of positive integers.
USAGE	This field is managed by GDK software and should not be modified by user programs.

time

DATA TYPE	GFQTIME
DESCRIPTION	When a record is submitted to the Queue File, it must contain the time it is to be processed or was processed, which is specified by this field.
VALUE	None.
USAGE	time (&(qrec->time)); /*for immediate processing*/

transmit_control

DATA TYPE	GFQINT
DESCRIPTION	This field is a 16-bit switch that controls various aspects of the Send operation. If all bits are set to 0, no special features will be included. Setting the first bit allows a cover page to be sent. Setting the second bit allows a header to be sent. Setting the third bit turns on the non-standard-facilities (NSF) field during transmission. This field must be specified when used in conjunction with the various controls.
VALUE	GFQUSE_COVERSHEET, GFQUSE_HEADER, GFQUSE_NSF, GFQUSE_OVERLAY_HEADER, GFQUSE_OVERLAY_HEADER_OR.
USAGE	qrec->transmit_control|= GFQUSE_HEADER; qrec->transmit_control|= GFQUSE_NSF;
SEE ALSO	nsf_field and fn_cover field descriptions.

user_id

DATA TYPE	GFQBYTE[GFQUSER_FIELD_SIZE]
DESCRIPTION	This 32-byte field indicates the submitter of the record, a network address, and a password. It can contain any alphanumeric characters; it also can be empty. Records may be submitted to the Queue File by specifying this field and using a function such as gfqSubmit. If the fax channel receives any routing information, it will post the results in as a coded field. For example: SYSOP;D=XXXX where “SYSOP” is the name generated by the communications program “;D” is a separator, and “XXXX” represents the DID (direct inward dialing) or DTMF (dual-tone multi-frequency) digits; or SYSOP;S:=YYYY... where “;S:” is a separator and “YYYY...” represents a

user_id

DATA TYPE	GFQBYTE[GFQUSER_FIELD_SIZE] received subaddress. Or ;P:=ZZZ… where ";P:" is a separator and "ZZZ…" represents a T.30 password.
VALUE	A character array of size [GFQUSER_FIELD_SIZE].
USAGE	strcpy (qrec->user_id,"John");
SEE ALSO	"Routing" section in Chapter 2.

5. Programming Models

GDK Subsystem

Dialogic has developed a fax subsystem optimized for mission critical, high volume, fax applications. CP Fax hardware channels are integrated through direct connection to the Public Switched Telephone Network (PSTN) or as a shared resource in a telephony bus environment such as Pulse Code Modulation Expansion Bus (PEB) or Signal Computing Bus (SCbus). All the details, critical timings, fax machine compatibility, on-line image conversions, High-Level Data-Link Control (HDLC) and signal quality measurements occur on-card via field upgradable, soft loadable firmware.

Architecturally, a GDK channel functions as a client-server to the host computer. Communication between the fax channel server and the host application client is through an operating-system dependent driver (the GDK system service dispatcher) and a GDK API messaging interface. This method differs greatly from traditional (Class 1 or Class 2) fax-data modems.

The GDK APIs provide fax communications in two ways — passive batch mode and runtime interactive mode. These two modes are described in more detail later in this chapter.

During fax transactions, the Dispatcher is a server to the fax channels executing proprietary remote procedure call requests for files, data, system time, or other host services. The application can participate in fax communication through event monitoring using the run-time API, or can ignore the details of the fax protocols by using the batch-programming model.

The GDK architecture shields the application from critical protocol timings, allowing the development of very high-density systems with inexpensive host chassis hardware. A fax transaction can withstand up to 30-second delay in host service. This delay can occur in Local Area Networks (LANs) and Wide Area Networks (WANs) used to connect mission critical, integrated fax solutions in businesses and service bureaus.

This architecture also protects the T.30 protocol from misuse in foreign countries where the fax protocol may be restricted by government approval requirements. Dialogic CP Fax products are approved for many countries.

Phases of a Fax Session

A facsimile session between two fax devices consists of five distinct phases (see Table 26). The ITU T.30 recommendation describes the interaction between two fax devices in more detail.

Table 26. Facsimile Session Phases

ITU T.30 Phase	Description	Fax Modem
Phase A — Call Setup	This phase establishes a call connection between the two devices, which includes dialing, call progress, answer and supervision (start the billing).	Calling fax device transmits CNG 1100 Hz tone, .5 second ON, 3 seconds OFF Answering fax device transmits CED 2100 Hz tone, 3 second duration
Phase B — Pre-Message Procedure	This phase consists of the mutual recognition of the fax devices (known as a handshake). It is a negotiation procedure that identifies their respective capabilities, identities and any non-standard facilities. This phase selects the session parameters for the message transmission including speed, compression and error correction.	V.21 mode, HDLC 300 bps
Phase C — Message Transmission	This phase is where the data is transmitted. The fax devices send/receive page files of the document, at the selected speeds performing T.4, T.6 and/or ECM as negotiated in Phase B.	v.17, v.29, v.27, v.33 14400, 12000, 9600, 7200,4800, 2400 bps optional HDLC (ECM mode)

ITU T.30 Phase	Description	Fax Modem
Phase D — Post-Message Procedure	After sending/receiving a page file of the document, a message exchange occurs between the fax devices indicating the success or failure to receive the page file, continue to next page file under same conditions, retrain to new conditions or to confirm the document transmission is complete.	V.21 mode, HDLC
Phase E — Call Release	In this phase the fax transaction is complete, good or failed, and the call is released (OFF-HOOK) terminating the billing and the connection.	(inactive)

Fax Programming Models

Dialogic CP Fax Series products define and support two programming models for facsimile communications — batch mode and run-time interactive models. The GFQ functions are used with the batch programming model, and the GRT functions are used with the run-time interactive programming model.

Batch Programming Model

The batch programming model relieves the developer of monitoring the fax transaction phases. This programming model is designed for applications such as automated transmission between two fax machines with no human intervention. The batch mode also minimizes the loading on the host computer.

A batch application cannot perform resource sharing or call switching. An additional limitation is that it is restricted to CP Fax hardware, which can perform direct call control (e.g. CP Fax cards with direct connection to the PSTN (analog boards or AEB connections, or direct control of signaling bits in digital environments such as PEB). As a result of the limitations of batch programming, developers generally prefer the interactive programming model, which is discussed later in this chapter.

NOTE: The batch programming model does **NOT** support the "transparent ISDN" or "transparent DM3 support" features.

In the batch programming model, the fax channel performs all five phases of a fax transaction. This API model can be used in the following types of applications:

- Fax broadcast
- E-mail fax gateways
- Two-call fax-on-demand applications

Batch mode programming deals with the submission and retrieval of *queue records* (Chapter 4). A queue record is simply a data structure that is stored in a database (the Queue File). This “database record” describes the fax transaction from start to finish.

For example, to send a fax, a queue record is filled in with the file name of the image and the phone number. The record is then submitted to the database using gfqSubmit( ) for processing. The GDK subsystem processes the fax transaction and updates the queue record with the status, duration, and completion time of the job. The completed record can then be retrieved by the application using gfqFindFirst( ) to update its local records.

This description demonstrates the simplicity of the batch mode model. Note, however, that the queue record is a comprehensive data structure, which provides

detailed transmission or receiving characteristics. Additionally, a completed queue record contains detailed measurements of line quality, fax machine characteristics, and call duration.

The level to which you program the queue record determines the sophistication of your fax application in the batch mode programming model. Conversions, fonts, headers, speeds, file transfers, retries, and special information exchanges are all available using the fields of the queue record.

NOTE: The batch programming model is an outdated paradigm with many limitations; Dialogic recommends the interactive programming model for most applications and environments.

GFQ APIs — Alphabetized List of the GFQ Functions

The following is an alphabetized list of the GFQ functions. These functions submit and retrieve transactions within the GDK Queue File (database). Any queue record that is not marked busy can be created, read, modified, or deleted from the Queue File by using the GFQ functions. For a detailed discussion of the Queue File's structure, please see Chapter 4.

Name:	void gfqClearRec (GFQRECORD *qrec);	
Inputs:	GFQRECORD *qrec	• A pointer to a queue record.
Outputs:	GFQRECORD *qrec	• A pointer to a queue record.
Returns:	None	
Includes:	gfq.h	

■ Description

The **gfqClearRec( )** function clears and initializes a record to the default settings listed in Table 27.

Table 27. Queue Record Default Values

Queue-Record Field	Default
qrec->time	time
qrec->notify	GFQNOTIFY_ONERROR
qrec->retry_strategy	GFQFULL_RETRY
qrec->retry_counter	3
qrec->retry_delay	30
qrec->rate	GFQMAX_RATE
qrec->protocol	GFQT30_PROTOCOL
qrec->number_calls	1
qrec->cd_timeout	30
qrec->source_type	GFQSINGLE_DOC

■ Example

```
#include "gfq.h"
GFQRECORD qrec;
gfqClearRec (&qrec);
```

Name:	int gfqFindFirst (GFQCHAR *gfqFileName, GFQRECORD *pQRec, GFQINT Link, GFQINT End, GFQCHAR *User);	
Inputs:	GFQCHAR *gfqFileName	• Name of Queue File access.
	GFQRECORD *pQRec	• Queue record buffer.
	GFQINT Link	• Link list to be read.
	GFQINT End	• The end from which to begin — GFQLIST_START or GFQLIST_END.
	GFQCHAR *User	• Valid queue record, if return is GFQSUCCESS.
Outputs:	GFQRECORD *pQRec	• Valid queue record, if return is GFQSUCCESS.
Returns:	GFQSUCCESS	• A valid record was found.
	GFQLIST_NOTFOUND	• The link specified was not valid.
	GFQLIST_EOF	• No records matching the search values were found.
	GFQRECORD_NOTFOUND	• List is empty.
Includes:	gfq.h	

■ Description

The **gfqFindFirst( )** function locates the first item in a list using the user ID, and the direction it was passed to determine which record is the first. That record is then read into the buffer pointed to by pQRec.

Example

```
#include "gfq.h"
#include "gfqpath.h"
int status;
GFQRECORD qrec;
GFQCHAR gfqFileName[GFQFILENAME_SIZE];
int i=0;
if (gfqSearch(GFQDIR_QUEUE, "gfax.$qu", gfqFileName)){
  printf("Error getting path to queue file.\n");
  exit(1);
}
status = gfqFindFirst(gfqFileName, &qrec, GFQPEND_LIST,
GFQLIST_START,"");
if (status == GFQSUCCESS){
    printf("Record %d on Pending list\n", ++i);
    printf("\t Phone = %s\n", qrec.phone_no);
    printf("\t Send file = %s\n", qrec.fn_send);
}
```

Name:	int gfqFindNext (GFQCHAR *gfqFileName, GFQRECORD *pQRec, GFQINT Direction, GFQCHAR *User)
Inputs:	GFQCHAR *gfqFileName • Name of queue file to access. GFQRECORD *pQRec • Queue record buffer. GFQINT Direction • Direction to read from current. Values are GDQREAD_FWD and GFQREAD_BWD GFQCHAR *User • User id string or “” for all.
Outputs:	None
Returns:	GFQSUCCESS • A valid record was found. GFQLIST_NOTFOUND • The link specified was not valid. GFQLIST_EOF • No records matching the search values were found. GFQRECORD_NOT_FOUND • The list is empty. GFQRECORD_KEY_ERROR • The key information in the queue record has been modified since it was retrieved. GFQRECORD_INV_READ_DIR • Direction specified was unrecognized.
Includes:	gfq.h

■ Description

The **gfqFindNext( )** function reads the next record in the specified direction, verifies the record passed in from the application has not changed, then it finds the next record in a list. Using the User parameter will limit the search of valid records.

■ Example

```
#include "gfq.h"
#include "gfqpath.h"
int status;
GFQRECORD qrec;
GFQCHAR gfqFileName[GFQFILENAME_SIZE];
int i=0;
if (gfqSearch(GFQDIR_QUEUE, "qfax.$qu", gfqFileName)){
  printf("Error getting path to queue file.\n");
  exit(1);
}
status = gfqFindFirst(gfqFileName, &qrec, GFQPEND_LIST,
GFQLIST_START,"");
while (status == GFQSUCCESS){
  printf("Record %d on Pending list\n", ++i);
  printf("\t Phone = %s\n", qrec.phone_no);
  printf("\t Send file = %s\n", qrec.fn_send);
  status = gfqFindNext(gfqFileName, &qrec, GFQREAD_FWD,"");
}
```

Name:	int gfqGetPath (int ft, char *fn, char *fullfn);	
Inputs:	int ft	• Selects the symbolic constant. See Table 27.
	char *fn	• A pointer to a file, such as the queue file, “gfax.$qu”, or a file to send, such as test001.tif.
	char *fullfn	• A pointer to a character buffer that is large enough to hold a fully qualified file path.
Outputs:	char *fullfn	• A pointer to a fully-qualified filename when a file does exist or may be created.
Returns:	GFQSUCCESS	• The function completed successfully; the “path” points to the desired file.
	GFQPATH_NOT_SET	• The file could not be found because the configuration information was not present. This is a fatal error.
	GFQPATH_INVALID	• The file could not be found because the configuration information was in error. This is a fatal error.
	GFQPATH_NO_FILE	• The file could not be found, but the configuration information seemed correct. The “path” points to the full filename where the file should exist.
	GFQPATH_BAD_TYPE	• The file type specified was invalid.
	GFQPATH_BAD_FILE	• The file specified is null.
Includes:	gfq.h path.h	

■ Description

The **gfqGetPath( )** function can be used to construct the path to a subdirectory where a file should be written, and can be used to locate the place to put a file that may not exist. It always returns a fully qualified filename, even if the file is not found. The function gfqSearchPath( ) should be used to locate a file that must exist.

gfqGetPath( ) uses the symbolic constants defined in Table 28. When a specific file type is sought, the function looks for an environment variable, such as those in the “Initial Search Target” column of Table 28. If an environment variable exists and is correct, the function uses it to create a full filename. If the environment variable was not specified, the function looks for an alternate environment variable or the current directory. If everything is proper, it returns a full filename. In the example that follows, GFQDIR_SEND is one of the basic file types (symbolic constants) defined. “path” in the example is a pointer to the full name of the file returned by gfqGetPath().

■ Example

```
#include "gfq.h"
#include "gfqpath.h"
char path[64];
if (gfqGetPath(GFQDIR_SEND, "test001.tif", path) == GFQSUCCESS)
  printf ("The path is %s\n", path);
```

Table 28. Symbolic Constants for gfqGetPath()

Symbolic Constant	Initial Search Target	Second Target	Meaning
GFQDIR_QUEUE	GFAXQ	GFAX	Locate the Queue File.
GFQDIR_UTILITY	GFAXU	GFAX	Locate the utility programs.
GFQDIR_SEND	GFAXS	current directory	Locate the send files.
GFQDIR_RECEIVE	GFAXR	current directory	Locate the received files.
GFQDIR_LOG	GFAXL	GFAX	Locate the log files.
GFQDIR_CONFIG	GFAXC	current directory	Locate the config files.

Name:	void gfqInsertOne (char *gfqFileName, GFQRECORD *qrec, int list);	
Inputs:	Char *gfqFileName	• A pointer to the name of the Queue File.
	GFQRECORD *qrec	• A pointer to the Queue File record to be inserted.
	int list	• The specified linked list of the Queue File.
Outputs:	None	
Returns:	GFQSUCCESS	• The record was added.
	GFQFILE_BUSY	• The Queue File is locked by another task.
	GFQFILE_CREATE_ERROR	• Unable to create the new Queue File.
	GFQFILE_INCOMPATIBLE	• The Queue File version is incompatible with this version of the GDK software.
	GFQFILE_NOTFOUND	• The Queue File was not found.
	GFQFILE_OPEN_ERROR	• Unable to open the Queue File.
	GFQRECORD_WRITE_ERROR	• Unable to create a new queue record.
	GFQLIST_NOTFOUND	• Unable to find the specified list.
Includes:	gfq.h	

■ Description

The **gfqInsertOne( )** function adds one record into the desired linked list of the Queue File. It is a general-purpose function that can be used to submit records to any linked list specified by the parameter “link.” Table 29 lists the names used with gfqInsertOne( ).

Each linked list in the Queue File is kept sorted, in descending order, by a time stamp and priority. This means that events to be processed “now” are at the end of the linked list and “future” events are at the beginning. A pointer to a record and a linked-list number are passed into this function. If the linked list is out of range, the function returns GFQLIST_NOTFOUND.

Table 29. List Names Used with gfqInsertOne()

Mnemonic	Meaning
GFQPEND_LIST	Pending List
GFQRECV_LIST	Received List
GFQSENT_LIST	Sent List
GFQCONV_LIST	Conversion List
GFQCTRL_LIST	Control List
GFQCPST_LIST	Control Done List

■ Example

```
#include "gfq.h"
#include "gfqpath.h"

GFQRECORD qrec;
char queuefile[128] = “\0”;
int status;

/* get fully qualified path to queue file */
if (gfqGetPath(GFQDIR_QUEUE, "gfax.$qu", queuefile) != GFQSUCCESS)
{
  fprintf(stderr, "GFAX environment variable not defined\n");
  exit(1);
}
```

```
/* initialize and fill in queue record */
...

status = gfqInsertOne(queuefile, &qrec, GFQPEND_LIST);
if (status != GFQSUCCESS)
{
  printf("gfqInsertOne to GFQPEND_LIST failed, %d\n", status);
  myErrorRoutine(status);
}
else
    printf("Queue Record successfully submitted to Pending
  List\n");
```

Name:	int gfqInsertPlist (GFQCHAR *gfqFileName, GFQRECORD *qrec, int list, GFQCHAR *phonelist);	
Inputs:	GFQCHAR *gfqFileName	• A pointer to the name of the Queue File.
	GFQRECORD *qrec	• A pointer to a queue record.
	int list	• The specified linked list of the Queue File.
	GFQCHAR *phonelist	• A pointer to the name of the file containing the list of phone numbers.
Outputs:	None	
Returns:	GFQSUCCESS	• The record(s) were inserted.
	GFQFILE_BUSY	• The Queue File is locked by another task.
	GFQFILE_CREATE_ERROR	• Unable to create the new Queue File.
	GFQFILE_INCOMPATIBLE	• The Queue File version is incompatible with this version of the GDK software.
	GFQFILE_NOTFOUND	• The Queue File was not found.
	GFQFILE_OPEN_ERROR	• Unable to open the Queue File.
	RECORD_WRITE_ERROR	• Unable to create a new queue record.
	GFQPHONE_FILE_ERROR	• The phone-list file could not be opened.
Includes:	gfq.h	

■ Description

The **gfqInsertPlist( )** function inserts one record into a list for every phone number in the phone list. It writes a queue record for each entry in the phone list into the specified list of the Queue File. The phone-list file is automatically opened, processed, and closed.

The structure of a record inside a phone list is shown below, and followed by the maximum length of each field:

Table 30. gfqInsertPlist Phone Number Record Structure

Field	Maximum Length
phone_number	20
rate	4
cd_timeout	4
last_name	20
first_name	12
company	20
category	10
class	10
voice	20

Each field must be delimited by a space. Each line in the file is a record and must be separated by a control line feed (CR LF).

The minimum requirement for a valid phone-list record is the phone_no field. If the rate and cd_timeout are missing, the values from the queue record are taken. If the rate specified in the queue record is different from the phone-list record, the lower of the two values is used. If the phone list is not specified (NULL or ""), gfqInsertPlist( ) calls gfqInsertOne( ).

Table 31 lists the names used with gfqInsertPlist( ).

The user is responsible for verifying the following:

- A phone-list file exists, and that it was correctly created.
- Fields of the queue record, such as cd_timeout and trans_rate, are set to their defaults.

Table 31. List Names Used with gfqInsertPlist()

Mnemonic	Meaning
GFQPEND_LIST	Pending List
GFQRECV_LIST	Received List
GFQSENT_LIST	Sent List
GFQCONV_LIST	Conversion List
GFQCTRL_LIST	Control List
GFQCPST_LIST	Control Done List

■ Example

```
#include "gfq.h"
#include "gfqpath.h"

GFQRECORD qrec;
char queuefile[128] = "\0";
char phonelist[128] =
"c:\\broadcst\\phonelst.txt";
int status;

/* get fully qualified path to queue file */
if (gfqGetPath(GFQDIR_QUEUE, "gfax.$qu", queuefile) != GFQSUCCESS)
{
    fprintf(stderr, "GFAX environment variable not defined\n");
    exit(1);
}
```

```
/* initialize and fill in queue record */
...

status = gfqInsertPlist(queuefile, &qrec, GFQPEND_LIST, phonelist);
if (status != GFQSUCCESS)
{
    printf("gfqInsertPlist failed, %d\n", status);
    myErrorRoutine(status);
}
else
        printf("Fax broadcast successfully submitted to Pending
    List\n");
```

Name:	int gfqPurgeAll (char *gfqFileName, GFQRECORD *qrec, int list, char *user);	
Inputs:	char *gfqFileName	• A pointer to the name of the Queue File.
	GFQRECORD *qrec	• A pointer to the queue record.
	int list	• A linked list in the Queue File.
	char *user	• A pointer to a user's identification.
Outputs:	None	
Returns:	GFQSUCCESS	• The purge was executed.
	GFQFILE_BUSY	• The Queue File is locked by another task.
	GFQFILE_CREATE_ERROR	• Unable to create the Queue File.
	GFQFILE_INCOMPATIBLE	• The Queue File version is incompatible with this version of the GDK software.
	GFQFILE_NOTFOUND	• The Queue File was not found.
	GFQFILE_OPEN_ERROR	• Unable to open the Queue File.
	GFQRECORD_ACTIVE	• The record is active; i.e., being processed by another task.
	GFQLIST_NOTFOUND	• The list is not a valid Queue File List.
Includes:	gfq.h	

■ Description

The **gfqPurgeAll( )** function purges from one of the Queue Files lists all records for a specific user that are not active or BUSY. If the user is NULL or "", all records on the list that are not busy are purged. If the purge succeeds, the function returns GFQSUCCESS, even when the list had no records.

To make disk space available, gfqPurgeAll( ) also automatically shrinks the Queue File to its minimum size of 78 bytes, leaving only the Queue File header, but under two conditions:

- Automatic shrinking has been allowed. When using the GFQRESET program, automatic shrinking can be enabled with the -r0 option after preallocating records for the Queue File.
- None of the linked lists contain records.

■ Example

```
#include "gfq.h"
#include "gfqpath.h"

GFQRECORD qrec;
char queuefile[128] = "\0";
int status;

/* get fully qualified path to queue file */
if (gfqGetPath(GFQDIR_QUEUE, "gfax.$qu", queuefile) != GFQSUCCESS)
{
  fprintf(stderr, "GFAX environment variable not defined\n");
  exit(1);
}

status = gfqPurgeAll(queuefile, &qrec, GFQSENT_LIST, Delete_Me");
if (status != GFQSUCCESS)
{
  printf("gfqPurgeAll Sending List failed, %d\n", status);
  myErrorRoutine(status);
}
else
  printf("All ‘Delete_Me’ records purged from Sending List\n");
```

Name:	int gfqPurgeOne (char *gfqFileName, GFQRECORD *qrec);	
Inputs:	char *gfqFileName;	• Pointer to the name of the Queue File.
	GFQRECORD *qrec	• Pointer to the queue record to be purged.
Outputs:	None	
Returns:	GFQSUCCESS	• The deletion was successful.
	GFQFILE_BUSY	• The Queue File is locked by another task.
	GFQFILE_CREATE_ERROR	• Unable to create the new Queue File.
	GFQFILE_INCOMPATIBLE	• The Queue File version is incompatible with this version of the GDK software.
	GFQFILE_NOTFOUND	• The Queue File was not found.
	GFQFILE_OPEN_ERROR	• Unable to open the Queue File.
	GFQRECORD_ACTIVE	• The record is active; that is, being processed by another task.
	GFQRECORD_KEY_ERROR	• Key values were changed.
Includes:	gfq.h	

■ Description

The **gfqPurgeOne( )** function deletes non-busy records from the Queue File. Before using this function, a queue record must have been successfully read using gfqFindFirst( ) or gfqFindNext( ). Both of these functions determines whether another task has changed the key values of the queue record (linked list, time, or priority). If the key values have been changed, gfqPurgeOne( ) returns the error GFQRECORD_KEY_ERROR. If a queue record is marked BUSY, the function returns the error GFQRECORD_ACTIVE.

Example

```
#include "gfq.h"
GFQRECORD qrec;
char gfqFileName[GFQFILENAME_SIZE];
int status;
strcpy (gfqFileName, "c:\\fax\\gfax.$qu");
if ((status = gfqFindFirst (gfqFileName, &qrec, GFQPEND_LIST,
GFQLIST_START,"")) == GFQSUCCESS)
if (qrec->user_id == "DELETE_ME")
  return (gfqPurgeOne (gfqFileName, &qrec));
```

Name:	int gfqSearch (int ft, char *fn, char *fullfn);	
Inputs:	int ft	• Select the file type (symbolic constant) listed in Table 27.
	char *fn	• The name of a user-specific file, such as “GFAX.$QU” or test001.tif.
Outputs:	char *fullfn	• A fully qualified filename of an existing file is returned only if the function is successful. A null string is returned if the function is not successful.
Returns:	GFQSUCCESS	• The function completed successfully; the “path” points to the desired file, and it does exist.
	GFQPATH_NOT_SET	• The file could not be found because the environment information was not present. This is a fatal error.
	GFQPATH_INVALID	• The file could not be found because the environment information was in error. This is a fatal error.
	GFQPATH_NO_FILE	• The file could not be found, but the environment information seemed correct. No filename is returned.
	GFQPATH_BAD_TYPE	• The file type specified was invalid.
Includes:	gfq.h gfqpath.h	

■ Description

The **gfqSearch( )** function returns a fully qualified filename. It verifies that a file of the given name and type does exist. It is used to locate existing GDK files, such as the Queue File and configuration files.

The environment variables for gfqSearch( ) are listed in Table 32. The symbolic constants used in this function are in the gfqpath.h file.

Table 32. Environment Variables for gfqSearch()

Symbolic Constant	Initial Search Target	Second Target	Meaning
GFQDIR_QUEUE	GFAXQ	GFAX	Locate the Queue File.
GFQDIR_UTILITY	GFAXU	GFAX	Locate the utility programs.
GFQDIR_SEND	GFAXS	current directory	Locate the send files.
GFQDIR_RECEIVE	GFAXR	current directory	Locate the received files.
GFQDIR_LOG	GFAXL	GFAX	Locate the log files.
GFQDIR_CONFIG	GFAXC	GFAX	Locate the configuration files.

■ Example

```
#include "gfq.h"
#include "gfqpath.h"
char path[64];
if ((status = gfqSearch (GFQDIR_QUEUE, "gfax.$qu", path)) ==
GFQSUCCESS)
{
  printf ("Queue file path is %s\n", path);
}
else
{
  printf ("gfqSearch failed, status = %d\n", status);
}
```

Name:	int gfqSubmit (char *gfqFilename, GFQRECORD *qrec);	
Inputs:	char *gfqFileName	• A pointer to the name of the Queue File.
	GFQRECORD *qrec	• A pointer to the queue record to be submitted.
Outputs:	None	
Returns:	GFQSUCCESS	• The record was added.
	GFQFILE_BUSY	• The Queue File is locked by another task.
	GFQFILE_CREATE_ERROR	• Unable to create the new Queue File.
	GFQFILE_INCOMPATIBLE	• The Queue File version is incompatible with this version of the GDK software.
	GFQFILE_NOTFOUND	• The Queue File was not found.
	GFQFILE_OPEN_ERROR	• Unable to open the Queue File.
	GFQRECORD_WRITE_ERROR	• Unable to create a new queue record.
Includes:	gfq.h gfqpath.h	

■ Description

The **gfqSubmit( )** function adds one record to the Pending List. Because this list is sorted in descending order by time stamp and priority, events to be processed in the future are at the beginning of the list and events to be processed "now" are at the end. Unlike gfqInsertOne( ), which can be used to submit records to *any* linked list, gfqSubmit( ) is used to submit queue records only to the Pending List.

The queue record is inserted into the Pending List, if the operation field of the record has the values between GFQDIAL_SEND and GFQLAST_OPERATION. Table 33 lists the values that gfqSubmit( ) automatically writes to queue-record fields.

Table 33. Values Written by gfqSubmit() to Queue Record Field

Field	Value
submission_time	The current time
record_control	Not marked busy
submission_retries	retry_counter
duration	0
status	0

■ Example

```
#include "gfq.h"
#include "gfqpath.h"

GFQRECORD qrec;
char queuefile[128] = "\0";
int status;

/* get fully qualified path to queue file */
if (gfqGetPath(GFQDIR_QUEUE, "gfax.$qu",
queuefile) != GFQSUCCESS)
{
  fprintf(stderr, "GFAX environment variable not defined\n");
  exit(1);
}
```

```
/* initialize queue record */
gfqClearRec(&qrec);

qrec.operation = GFQDIAL_SEND;
strcpy(qrec.fn_send, "c:\\fax\\test001.tif");
strcpy(qrec.phone_no, "1-408-555-1212");

status = gfqSubmit(queuefile, &qrec);
if (status != GFQSUCCESS)
{
  printf("gfqSubmit error %d\n", status);
  myErrorRoutine(status);
}
else
  printf("fax submitted successfully\n");
```

Name:	int gfqSubmitPlist (char *gfqFileName, GFQRECORD *qrec, char *phonelist);	
Inputs:	char *gfqFileName	• A pointer to the name of the Queue File.
	GFQRECORD *qrec	• A pointers to a queue record.
	char *phonelist	• A pointer to the name of the file containing the list of phone numbers.
Outputs:	None	
Returns:	GFQSUCCESS	• The submission was successful.
	GFQFILE_BUSY	• The Queue File is locked by another task.
	GFQFILE_CREATE_ERROR	• Unable to create the new Queue File.
	GFQFILE_INCOMPATIBLE	• The Queue File version is incompatible with this version of the GDK software.
	GFQFILE_NOTFOUND	• The Queue File was not found.
	GFQSUCCESS	• Unable to open the Queue File.
	GFQFILE_BUSY	• Unable to create a new queue record.
	GFQFILE_CREATE_ERROR	• The phone list cannot be opened.
Includes:	gfq.h	

Description

The **gfqSubmitPlist( )** function submits a fax for each entry in a phone list.

This function starts fax broadcasting by submitting records to the Pending List for an entire phone list. The phone-list file is automatically opened, processed, and closed. gfqSubmitPlist( ) writes a record for each phone number from the phone list into the Pending List of the Queue File.

The operation field of the queue record determines the action to be taken on each record submitted, such as Dial and Send, Dial and Receive, or Answer Immediately. The queue record is inserted into the Pending List if the operation field of the record has the values between GFQDIAL_SEND and GFQLAST_OPERATION. If the operation for a record is appropriate, gfqSubmitPlist( ) calls gfqInsertPlist( ) and passes the record, specifying the Pending List as the target. The caller should verify that the phone-list file exists and is correctly formatted. See gfqInsertPlist( ) for the phone-record format.

These fields of the queue record are set to their defaults in the phone list: 30 seconds for cd_timeout and GFQMAX_RATE bps for trans_rate.

Example

```
#include "gfq.h"
#include "gfqpath.h"

GFQRECORD qrec;
char queuefile[128] = "\0";
char phonelist[128] = "c:\\broadcst\\phonelst.txt";
int status;

/* get fully qualified path to queue file */
if (gfqGetPath(GFQDIR_QUEUE, "gfax.$qu", queuefile)
!= GFQSUCCESS)
{
  fprintf(stderr, "GFAX environment variable not defined\n");
  exit(1);
}
```

```
/* initialize queue record */
gfqClearRec(&qrec);
qrec.operation = GFQDIAL_SEND;
strcpy(qrec.fn_send, "c:\\faxsend\\broadcst.tif");
status = gfqSubmitPlist(queuefile, &qrec, phonelist);
if (status != GFQSUCCESS)
{
  printf("gfqSubmitPlist error %d\n", status);
  myErrorRoutine(status);
}
else
  printf("fax broadcast submitted successfully\n");
```

Interactive Programming Model

The interactive programming model, which is recommended for fax-only applications, separates and provides distinct control at all phases of a fax transaction. In general, the call setup and call release phases are handled by the application using other telephony resources in the system such as network interface cards, voice boards or PBX APIs. The fax channel in this configuration only performs the pre-message, message transmission and post-message procedures (Phases B, C and D).

In this model, the application must monitor the call and decide when to start the fax resource. In contrast to the batch mode, the application can continue to monitor the call through the entire fax transaction, electing to change session parameters, interrupt the call or just abort all together. The monitoring activity is accomplished by examining “Events” — distinct points in the fax protocol where the fax channel notifies the application a fax event has occurred.

This programming model is a powerful feature of the GDK system but requires more knowledge of fax, more host computer resources and has timing requirements that are not present in the batch mode model. Figure 3 illustrates the GRT event notification phases.

NOTE: The TESTFAX sample code, which uses the interactive programming model, is located on the GDK product CD-ROM.

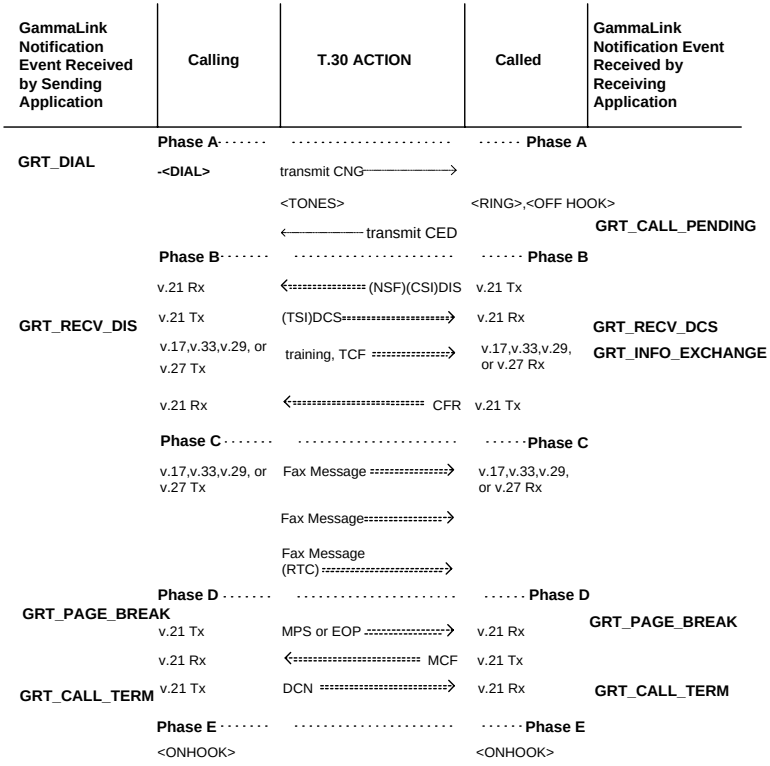

Figure 3. GRT Events

The GDK interactive, runtime API (GRT) events shown in Table 34 notify the fax application of the fax transaction's progress and provide opportunity for the application to get information generated during the transaction and, optionally, alter the course of the call based on that information.

Each event can be "armed" in one of two ways: by using the No Response Required method to simply receive notification, or using the Armed-Requires Response method so the application must respond to the event. In the case of response required, the application has choices such as GRT_CONTINUE, GRT_END_CALL, or use a GRT_QREC. For the case of responding with a new

queue record, see Chapter 4 for detailed information on queue record fields and their effects.

A simple interactive fax session is outlined below:

- initialize the run-time interface (grtInit( ))
- START
- establish the starting parameters for the fax transaction by programming a queue record (see Chapter 4)
- establish a connection to a fax machine (by dialing or answering) with a telephony network interface card in your system (i.e., analog, T1, E1, ISDN, etc.)
- start the fax session (grtSubmitFax( ))
- monitor for call completion (grtGetEvent( ))
- process event (grtProcessCallTermEvent( ))
- another call? go to START
- program complete (grtStop( ))

Sample GRT Applications

The GRT APIs provide multi-threaded support on a per-channel basis. An application can assign threads of control for one or more fax channels dynamically. A channel can only be controlled by one GRT thread.

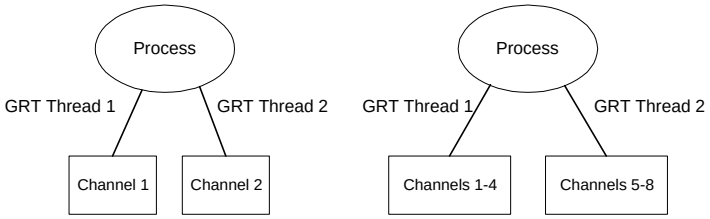

Figure 4. GRT API Structure

A simple GRT application consists of three major components: initialization, polling for an event, and termination.

Initialization

A GRT thread is initialized using the grtInit( ) function. This function accepts a range of channels, the same thread can control several channels. The function can be called multiple times in the same thread, as long as a GRT thread doesn't already control the channel range given.

NOTE: grtInit( ) will not create a new thread — it initializes existing internal structures that maintain GRT state information. A C function must be called to create a new thread.

Here is an example of how to initialize the first two fax channels for processing GRT_CALL_TERM events:

```
#include "genra.h"
int status;
status = grtInit(1,2,GRT_CALL_TERM_ENABLE);
```

Polling for an Event

Polling is used to check event status to determine whether an event has occurred. To check for an event, use the grtGetEvent( ) function. This function checks the channel range for a given event.

Here is an example of how to poll for a GRT event on the first two channels in a fax system:

```
GRT_EVENT event;

for(;;)
{
  if (grtGetEvent(1,2,&event) == GRT_SUCCESS)
    break;
  Sleep(1000L);
}
```

Termination

GRT API processing is terminated using the grtStop( ) function. This function stops any GRT processing in the given range of channels. This function allows for dynamic allocation of channels and threads as the system load changes.

The following is an example of how to stop GRT processing on channels 1 and 2 in a fax system:

```
#include "genra.h"
int status;
status = grtStop(1,2);
printf("status = %d\n",status);
```

NOTE: grtStop() must be called before exiting the application. Otherwise, grtInit() for the same channel range will return error code 103.

Advanced GRT Applications

Advanced GRT applications have an additional component: responding to an event (i.e. grtRespond, grtRespondEndCall, grtRespondContinue, grtRespondQueueRec). Responding to an event allows the application to change the default action for the fax channel. The fax channel default action for each event, listed in Table 34, is to continue the fax transaction through each phase, until GRT_CALL_TERM (Phase E), then post the completed queue record to the queue file.

If an application wants to change the default action of an event, event notification and event response enable must be specified for that event when initializing the GRT interface for the fax channel. At each fax channel event notification, if event response is enabled, the application can respond to end the call (grtRespondEndCall) or to continue the call (grtRespondContinue).

For the GRT_CALL_PENDING event, the application has an additional response available (grtRespondQueueRec), with which the characteristics of the fax transmission can be changed through the submission of another queue record.

For example, turn-around polling can be accomplished by responding to the GRT_CALL_PENDING event with a queue record whose operation field is set to the GFQANSWER_RECEIVE_SEND value. Other queue record operation field values are listed in Chapter 4.

GRT API Data Structures

An alphabetized list of the GRT API data structures appears in the following section.

Name: typedef struct {
int chan;
int event_type;
char pdata[GRT_MESSAGE];
int num_bytes;
} GRT_EVENT;

Includes: genra.h

■ Description

The **GRT_EVENT** structure is used to store an event that has occurred on a given channel. The GRT_EVENT data structure contains the following information:

1. The channel on which the event occurred.
2. The type of event that has occurred. (See Table 34.)
3. Data that is associated with the event. (See Table 34.)
4. Number of bytes of event data.

The definition of the GRT_EVENT structure follows:

Table 34. Events and Data Associated with GRT_EVENT

Mnemonic	Associated Data
GRT_CALL_TERM	Queue record
GRT_INFO_EXCHANGE	GRT_INFO_DATA
GRT_DIAL	20 character null-terminated string
GRT_RECV_DCS	Unsigned char array of size DCS_LENGTH
GRT_CALL_PENDING	None
GRT_RECV_DIS	Unsigned char array of size DIS_LENGTH
GRT_PAGE_BREAK	None

Name:
```
typedef struct {
  unsigned char rcsid[22];
  unsigned char nsf[226];
} GRT_INFO_DATA;
```
Includes: genra.h

■ Description

The **GRT_INFO_DATA** data structure stores event information for GRT_INFO_EXCHANGE( ). The structure contains the following information:

1. The Customer Subscriber Identification (CSID) of the receiving fax machine. The CSID is the string of characters that identify the remote fax machine (this string is usually the phone number of the remote fax machine).
2. The NSF data. NSF data is optional information that can be sent and received by fax machines.

Name:
```
typedef struct {
  int response_type;
  char pdata[GRT_MESSAGE]; int num_bytes;
} GRT_RESPONSE;
```
Includes: genra.h

■ Description

The **GRT_RESPONSE** structure stores information that is used to send a response to a fax channel once an event has occurred. The structure contains the following information:

- Type of response
- Data associated with the response
- Number of bytes of response data

■ Response Types

- GRT_END_CALL

- GRT_CONTINUE
- GRT_QREC

Name:	int grtGetEvent (int start_chan, int end_chan, GRT_EVENT *event)	
Inputs:	int start_chan	• Integer representing the first channel to control.
	int end_chan	• Integer representing the last channel to control.
	GRT_EVENT *event	• Pointer to GRT_EVENT.
Outputs:	GRT_EVENT *event	• Pointer to GRT_EVENT.
Returns:	GRT_SUCCESS	• Successful.
	GRT_INVALID_START_CHANNEL	• Starting channel is out of range.
	GRT_INVALID_END_CHANNEL	• Ending channel is out of range.
	GRT_NO_DISPATCHER	• Dispatcher is not running.
	GRT_NO_EVENT	• No event has occurred.
	GRT_UNKNOWN_EVENT	• Unknown event polled from board.
	GRT_NOT_INIT	• The specified channel has not been grtInit by any process or thread.
	GRT_ANOTHER_THREAD	• The specified channel has been grtInit by another process or thread (not this one).
	GRT_CONTROL_BUSY	• Timed out while waiting to set the mutex on the control record. Another thread failed to release the mutex. Retry.
Includes:	genra.h	

■ Description

The **grtGetEvent( )** function checks if an event has occurred on any channel in the range of channels given. The function returns after the first event occurs and the other channels are not checked.

■ Example

```
#include "genra.h"

GRT_EVENT event;
int status, chan = 1;

/* block for any event */
while ((status = grtGetEvent(chan, chan, &event)) == GRT_NO_EVENT)
       Sleep(1000L);
```

Name:	int grtInit (int start_chan, int end_chan, int attributes)	
Inputs:	int start_chan	• Integer representing the first channel to control.
	int end_chan	• Integer representing the last channel to control.
	int attribute	• Attributes associated with the given channel range. The attributes include programming model, event notification and response.
Outputs:	None	
Returns:	GRT_SUCCESS	• Successful.
	GRT_INVALID_START_CHANNEL	• Starting channel is out of range.
	GRT_INVALID_END_CHANNEL	• Ending channel is out of range.
	GRT_NO_DISPATCHER	• Dispatcher is not running.
	GRT_ALREADY_CONTROLLED	• If any channel in the range is controlled by another thread.
	GRT_CONTROL_BUSY	• Semaphore could not be locked.
	GRT_UNLOCK_FAIL	• Semaphore could not be unlocked.

GRT_LOCK_FAILED	• Semaphore could not be locked.
GRT_PROCESS_ATTRIBUTE_FAIL	• Could not enable notification or response on the fax channel.
GRT_OPEN_FAIL	• Could not open notification or response pipe to fax channel.

Includes: genra.h

■ Description

The **grtInit( )** function initializes the GRT interface. To initialize internal data structure with the proper value to control the given range of channels.

This function can be called multiple times in a thread as long as different ranges of channels are used.

These are the attributes that are currently supported.

- GRT_CALL_TERM_ENABLE
- GRT_INFO_EXCHANGE_ENABLE
- GRT_RECV_DIS_ENABLE
- GRT_RECV_DCS_ENABLE
- GRT_PAGE_BREAK_ENABLE
- GRT_DIAL_ENABLE
- GRT_CALL_PENDING_ENABLE
- GRT_CALL_TERM_RESPONSE_ENABLE
- GRT_INFO_EXCHANGE_RESPONSE_ENABLE
- GRT_CALL_PENDING_RESPONSE_ENABLE

Behavior

grtInit() has been enhanced to prevent automatic posting of the completed queue record to the queue file. This action is usually not necessary as most applications also receive a copy of the same queue record when the GRT_CALL_TERM_ENABLE attribute is specified in **grtInit().**

However, you can restore the ability of the GDK to automatically post the completed queue record to the queue file with the following **gfdRemoteRequest()** command as demonstrated in the code fragment:

```
#include <genra.h>
       ...
int grtAttributes = GRT_CALL_TERM_ENABLE;
int chan = 1, status, rresult;
       ...
status = grtInit( chan, chan, grtAttributes );
if ( GRT_SUCCESS == status )
{
   // allow posting of results to queue file
   status = gfdRemoteRequest(
                  1,
                  chan,
                  GFXRTACTION,
                  GFXRTBP_CALLTERM,
                  GFXRT_CONTINUE,
                  NULL,
                  0,
                  &rresult
                );

   if ( 0 == status ) {
      printf( "Success: Posting enabled!\n" );
   }
}
       ...
```

The gfdRemoteRequest command must be issued on a per-channel basis.

Example

```
#include "genra.h"

int status, chan = 1;

/* initialize GRT pipes interface for fax
channel 1 */
status = grtInit(chan,chan,GRT_CALL_TERM_ENABLE);
if (status != GRT_SUCCESS)
{
    fprintf(stderr, "[%2d] grtInit failed\n", chan);
    myErrorRoutine(status);
}
```

Name:	int grtProcessCallTermEvent (GRT_EVENT *event, GFQRECORD *qrec)	
Inputs:	GRT_EVENT *event	• Pointer to an event record.
	GFQRECORD *qrec	• Pointer to a queue record.
Outputs:	GFQRECORD *qrec	• Pointer to a queue record.
Returns:	GRT_SUCCESS	• Successful.
	GRT_UNKNOWN_EVENT	• Otherwise.
Includes:	genra.h	

■ Description

The **grtProcessCallTermEvent( )** function processes a call term event.

This function is a high-level function that provides the data that is associated with a GRT_CALL_TERM event to the user. The data will be a queue record. The function returns GRT_SUCCESS if the event is a GRT_CALL_TERM and the queue record pointer points to valid queue record data.

■ Example

```
#include "gfq.h"
#include "genra.h"
GRT_EVENT event;
GFQRECORD qrec;
int status, chan = 1;

/* block for any event */
while ((status = grtGetEvent(chan, chan, &event)) == GRT_NO_EVENT)
   Sleep(1000L);
if (event.event_type == GRT_CALL_TERM)
{
   if (status = grtProcessCallTermEvent(&event, &qrec))
      {
         printf("[%2d] grtProcessCallTermEvent error"
                " %d\n", event.chan, status);
         return -1;
      }
}
```

Name:	int grtProcessDialEvent (GRT_EVENT *event, char *dial_string)	
Inputs:	GRT_EVENT *event	• Pointer to an event record.
	char *dial_string	• Pointer to a 20-character string.
Outputs:	char *dial_string	• Pointer to a 20-character string.
Returns:	GRT_SUCCESS	• Successful.
	GRT_UNKNOWN_EVENT	• Otherwise.
Includes:	genra.h	

■ Description

The **grtProcessDialEvent( )** function processes a dial event.

This function is a high-level function that provides the data that is associated with a GRT_DIAL event to the user. The data will be a 20-character string representing the dialing string. The function returns GRT_SUCCESS if the event is a GRT_DIAL event and dial_string points to a valid dialing string. If the event is not GRT_DIAL, then the dial_string points to invalid data.

■ Example

```
#include "genra.h"

GRT_EVENT event;
char dialstring[20] = "\0";
int status, chan = 1;

/* block for any event */
while ((status = grtGetEvent(chan, chan, &event)) == GRT_NO_EVENT)
   Sleep(1000L);
```

```
if (event.event_type == GRT_DIAL)
{
   if (status = grtProcessDialEvent(&event, dialstring))
       {
          fprintf(stderr, "[%2d] grtProcessDialEvent"
                  " failed: %d\n", event.chan, status);
           return -1;
       }

    printf("[%2d] completed dialing to \"%s\"", event.chan,
dialstring);
}
```

Name:	int grtProcessInfoEvent (GRT_EVENT *event, GRT_INFO_DATA *info)	
Inputs:	GRT_EVENT *event	• Pointer to an event record.
	GRT_INFO_DATA *info	• Pointer to GRT_INFO_DATA structure.
Outputs:	GRT_INFO_DATA *info	• Pointer to GRT_INFO_DATA structure.
Returns:	GRT_SUCCESS	• Successful.
	GRT_UNKNOWN_EVENT	• Otherwise.
Includes:	genra.h	

■ Description

The **grtProcessInfoEvent( )** function processes an info exchange event.

This function is a high-level function that provides the data that is associated with a GRT_INFO_EXCHANGE event to the user. The data is a structure of type GRT_INFO_DATA. The function returns GRT_SUCCESS if the event is a GRT_INFO_EXCHANGE event and info points to a valid GRT_INFO_DATA structure.

■ Example

```
#include "genra.h"

GRT_EVENT event;
GRT_INFO_DATA info_data;
int status, chan = 1;

/* block for any event */
while ((status = grtGetEvent(chan, chan, &event))
== GRT_NO_EVENT)
      Sleep(1000L);
```

```
if (event.event_type == GRT_INFO_EXCHANGE)
{
   if (status = grtProcessInfoEvent(&event, &info_data))
      {
         fprintf(stderr, "[%2d] grtProcessInfoEvent"
                 " failed: %d\n", event.chan, status);
         return -1;
      }

      printf("[%2d] INFO EXCHANGE Data:\n"
             "\tRCSID:\t\"%s\"\n\tNFS:\t\"%s\"",
             event.chan, info_data.rcsid, info_data.nsf);
}
```

Name:	int grtProcessRecvDCSEvent (GRT_EVENT *event, unsigned char *dcs)	
Inputs:	GRT_EVENT *event	• Pointer to an event record.
	unsigned char *dcs	• Pointer to an array of unsigned char of size DCS_LENGTH.
Outputs:	unsigned char *dcs	• Pointer to an array of unsigned char of size DCS_LENGTH.
Returns:	GRT_SUCCESS	• Successful.
	GRT_UNKNOWN_EVENT	• Otherwise.
Includes:	genra.h	

■ Description

The **grtProcessRecvDCSEvent( )** function processes a received DCS event.

This function is a high-level function that provides the data that is associated with a GRT_RECV_DCS event to the user. The data is an array of unsigned characters representing the received DCS. The function returns GRT_SUCCESS if the event is a GRT_RECV_DCS event and DCS_frame points to valid DCS data. If the event is not GRT_RECV_DCS, then the DCS_frame points to invalid data.

■ Example

```
#include "genra.h"

GRT_EVENT event;
unsigned char dcs[DCS_LENGTH] = "\0";
const int count = sizeof(dcs)/sizeof(*(dcs));
int status, chan = 1;

/* block for any event */
while ((status = grtGetEvent(chan, chan, &event))
== GRT_NO_EVENT)
      Sleep(1000L);

if (event.event_type == GRT_RECV_DCS)
{
    if(status=grtProcessRecvDCSEvent(&event, dcs))
```

```
        {
            fprintf(stderr, "[%2d] grtProcessRecvDCSEvent failed:
                    %d\n", event.chan, status);
            return -1;
        }

        printf("[%2d] DCS received:", event.chan);
        for (i = 0; i < count; ++i)
              fprintf(stdout, "%02x ", dcs[i]);
        fprintf(stdout, "\n");
}
```

Name:	int grtProcessRecvDISEvent (GRT_EVENT *event, unsigned char *dis)	
Inputs:	GRT_EVENT *event	• Pointer to an event record.
	unsigned char *dis	• Pointer to an array of unsigned char of size DIS_LENGTH.
Outputs:	unsigned char *dis	• Pointer to an array of unsigned char of size DIS_LENGTH.
Returns:	GRT_SUCCESS	• Successful.
	GRT_UNKNOWN_EVENT	• Otherwise.
Includes:	genra.h	

■ Description

The **grtProcessRecvDISEvent( )** function processes a received DIS event.

This function is a high-level function that provides the data that is associated with a GRT_RECV_DIS event to the user. The data is an array of unsigned characters representing the received DIS. The function returns GRT_SUCCESS if the event is a GRT_RECV_DIS event and DIS_frame points to valid DIS data. If the event is not GRT_RECV_DIS, then the DIS_frame points to invalid data.

■ Example

```
#include "genra.h"

GRT_EVENT event;
unsigned char dis[DIS_LENGTH] = "\0";
const int count = sizeof(dis)/sizeof(*(dis));
int status, i, chan = 1;

/* block for any event */
while ((status = grtGetEvent(chan, chan, &event))
== GRT_NO_EVENT)
      Sleep(1000L);

if (event.event_type == GRT_RECV_DIS)
{
    if(status=grtProcessRecvDISEvent(&event, dis))
```

```
    {
       fprintf(stderr, "[%2d] grtProcessRecvDISEvent failed:"
               " %d\n", event.chan, status);
       return -1;
     }

     printf("[%2d] DIS received:", event.chan);
     for (i = 0; i < count; ++i)
            fprintf(stdout, "%02x ", dis[i]);
     fprintf(stdout, "\n");
}
```

Name:	int grtRespond (GRT_EVENT *event, GRT_RESPONSE *response)	
Inputs:	GRT_EVENT *event	• Pointer to an event structure.
	GRT_RESPONSE *response	• Pointer to a response structure.
Outputs:	None	
Returns:	GRT_SUCCESS	• Successful.
	GRT_ANOTHER_THREAD	• Channel is controlled by another thread.
	GRT_INVALID_RESPONSE	• Type of response to be sent is invalid.
	GRT_RESPONSE_FAIL	• Send of response message failed.
	GRT_INVALID_CHAN	• Channel that event occurred upon is not the same channel the response is being sent to.
	GRT_CONTROL_BUSY	• The semaphore could not be set.
	GRT_UNLOCK_FAIL	• The semaphore could not be released.
	GRT_LOCK_FAILED	• The semaphore could not be set.
	GRT_NOT_INIT	• The specified channel has not been grtInit'ed.
	GFD_BROKEN_PIPE	• The response pipe is broken.
	GFD_INVALID_HANDLE	• The response pipe is corrupted.
Includes:	genra.h	

■ Description

The **grtRespond( )** function responds to an event.

This function responds to the given event. The response structure must have the response_type (GRT_CONTINUE, GRT_END_CALL, etc.) initialized before this function is called.

■ Example

```
#include "genra.h"

GRT_EVENT event;
GRT_RESPONSE response;
int status, chan = 1;
int grtAttributes = GRT_CALL_TERM_ENABLE |
GRT_CALL_TERM_RESPONSE_ENABLE;

/* initialize GRT pipes interface for fax channel 1 */
status = grtInit(chan, chan, grtAttributes);
if (status != GRT_SUCCESS)
{
   fprintf(stderr, "[%2d] grtInit failed, %d\n", chan, status);
   return -1;
}

/* submit queue record */
...

/* block for any event */
while ((status = grtGetEvent(chan, chan, &event))
== GRT_NO_EVENT)
      Sleep(1000L);

if (event.event_type == GRT_CALL_TERM)
{
    /* fill in response structure */
    response.response_type = GRT_END_CALL;

    /* respond to event (do not post queue record to file) */
    if (status = grtRespond(&event, &response))
    {
       printf("[%2d] grtRespond error %d\n", event.chan, status);
       return -1;
    }
}
```

Name:	int grtRespondContinue (GRT_EVENT *event)	
Inputs:	GRT_EVENT *event	• Pointer to an event record.
Outputs:	None	
Returns:	GRT_SUCCESS	• Successful.
	GRT_ANOTHER_THREAD	• Channel is controlled by another thread.
	GRT_RESPONSE_FAIL	• Response to event failed.
	GRT_CONTROL_BUSY	• The semaphore could not be set.
	GRT_UNLOCK_FAIL	• The semaphore could not be released.
	GRT_LOCK_FAILED	• The semaphore could not be set.
	GRT_NOT_INIT	• The specified channel has not been grtInit'ed.
	GFD_BROKEN_PIPE	• The response pipe is broken.
	GFD_INVALID_HANDLE	• The response pipe is corrupted.
Includes:	genra.h	

■ Description

The **grtRespondContinue( )** function responds to an event with a continue message. This function responds to the given event with a GRT_CONTINUE response message.

■ Example

```
#include "genra.h"

GRT_EVENT event;
int status, chan = 1;
int grtAttributes = GRT_CALL_TERM_ENABLE |
GRT_CALL_TERM_RESPONSE_ENABLE;

/* initialize GRT pipes interface for fax
channel 1 */
status = grtInit(chan, chan, grtAttributes);
if (status != GRT_SUCCESS)
{
   fprintf(stderr, "[%2d] grtInit failed, %d\n", chan, status);
   return -1;
}

/* submit queue record */
...

/* block for any event */
while ((status = grtGetEvent(chan, chan, &event))
== GRT_NO_EVENT)
      Sleep(1000L);

if (event.event_type == GRT_CALL_TERM)
{
    /* respond to event (do default action, post queue record to
       file) */
   if (status = grtRespondContinue(&event))
   {
      printf("[%2d] grtRespondContinue error" " %d\n",
             event.chan, status);
      return -1;
   }
}
```

Name:	int grtRespondEndCall (GRT_EVENT *event)	
Inputs:	GRT_EVENT *event	• Pointer to an event record.
Outputs:	None	
Returns:	GRT_SUCCESS	• Successful.
	GRT_ANOTHER_THREAD	• Channel is controlled by another thread.
	GRT_RESPONSE_FAIL	• Response to event failed.
	GRT_CONTROL_BUSY	• The semaphore could not be set.
	GRT_UNLOCK_FAIL	• The semaphore could not be released.
	GRT_LOCK_FAILED	• The semaphore could not be set.
	GRT_NOT_INIT	• The specified channel has not been grtInit'ed.
	GFD_BROKEN_PIPE	• The response pipe is broken.
	GFD_INVALID_HANDLE	• The response pipe is corrupted.
Includes:	genra.h	

■ Description

The **grtRespondEndCall( )** function responds to an event with an end call message. This function responds to the given event with a GRT_END_CALL response message.

Example

```
#include "genra.h"

GRT_EVENT event;
int status, chan = 1;
int grtAttributes = GRT_RECV_DCS_ENABLE |
GRT_RECV_DCS_RESPONSE_ENABLE;

/* initialize GRT pipes interface for fax channel 1 */
status = grtInit(chan, chan, grtAttributes);
if (status != GRT_SUCCESS)
{
   fprintf(stderr, "[%2d] grtInit failed, %d\n", chan, status);
   return -1;
}

/* submit queue record */
...

/* block for any event */
while ((status = grtGetEvent(chan, chan, &event))
== GRT_NO_EVENT)
      Sleep(1000L);

if (event.event_type == GRT_RECV_DCS)
{
   /* respond to event (disconnect call after receiving DCS) */
      if (status = grtRespondEndCall(&event))
      {
          fprintf(stderr, "[%2d] grtRespondEndCall"
                  " failed, %d", event.chan, status);
          return -1;
      }
}
```

Name:	int grtRespondQueueRec (GRT_EVENT *event, GFQRECORD *qrec)	
Inputs:	GRT_EVENT *event	• Pointer to an event record.
	GFQRECORD *qrec	• Pointer to a queue record.
Outputs:	None	
Returns:	GRT_SUCCESS	• Successful.
	GRT_ANOTHER_THREAD	• Channel is controlled by another thread.
	GRT_RESPONSE_FAIL	• Response to event failed.
	GRT_CONTROL_BUSY	• The semaphore could not be set.
	GRT_UNLOCK_FAIL	• The semaphore could not be released.
	GRT_LOCK_FAILED	• The semaphore could not be set.
	GRT_NOT_INIT	• The specified channel has not been grtInit'ed.
	GFD_BROKEN_PIPE	• The response pipe is broken.
	GFD_INVALID_HANDLE	• The response pipe is corrupted.
Includes:	genra.h	

■ Description

The **grtRespondQueueRec( )** function responds to an event with a queue record. This function responds to the given event by sending a queue record to the channel on which the event occurred.

■ Example

```
#include "gfq.h"
#include "genra.h"

GRT_EVENT event;
GFQRECORD qrec;
int status, chan = 1;
int grtAttributes = GRT_CALL_PENDING_ENABLE |
GRT_CALL_PENDING_RESPONSE_ENABLE;

/* initialize GRT pipes interface for fax channel 1 */
status = grtInit(chan, chan, grtAttributes);
if (status != GRT_SUCCESS)
{
   fprintf(stderr, "[%2d] grtInit failed, %d\n", chan, status);
   return -1;
}

/* initialize, fill in, and submit queue record */
...

/* block for any event */
while ((status = grtGetEvent(chan, chan, &event))
== GRT_NO_EVENT)
      Sleep(1000L);

if (event.event_type == GRT_CALL_PENDING)
{
    /* respond to event */
    /* perform ANSWER-RECEIVE-SEND operation (turn-around polling)
*/
    if (status=grtRespondQueueRec(&event, &qrec))
    {
       fprintf(stderr, "[%2d] grtRespondQueueRec"
              " failed, %d", event.chan, status);
       return -1;
    }
}
```

Name:	int grtStop (int start_chan, int end_chan)	
Inputs:	int start_chan	• Starting channel in range of channels.
	int end_chan	• Ending channel in range of channels.
Outputs:	None	
Returns:	GRT_SUCCESS	• Successful.
	GRT_INVALID_START_CHANNEL	• Starting channel is out of range.
	GRT_INVALID_END_CHANNEL	• Ending channel is out of range.
	GRT_NO_DISPATCHER	• Dispatcher is not running.
	GRT_ANOTHER_THREAD	• Another thread controls a channel in the range given.
	GRT_CONTROL_BUSY	• The semaphore could not be locked.
	GRT_UNLOCK_FAIL	• The semaphore could not be unlocked.
	GRT_LOCK_FAILED	• The semaphore could not be locked.
	GRT_PROCESS_ATTRIBUTE_FAIL	• Could not enable notification or response on the fax channel.
	GRT_CLOSE_FAIL	• Could not close notification or response pipe to fax channel.
Includes:	genra.h	

■ Description

The **grtStop( )** function stops the GRT API from managing the fax channels. This function stops control of the given range of channels by the current thread of execution. Allowing starting and stopping of the GRT allows a thread to add or delete channels that it controls dynamically.

■ Example

```
#include "genra.h"

int status, chan = 1;

if (status = grtStop(chan, chan))
{
   fprintf(stderr, "[%2d] grtStop failed\n",  chan);
   myErrorRoutine(status);
}
```

Name:	int grtSubmitFax (int chan, GFQRECORD *qrec)	
Inputs:	int chan	• Channel on which fax is to be sent.
	GFQRECORD *qrec	• Pointer to a queue record.
Outputs	none	
Returns:	GRT_SUCCESS	• Successful.
	GRT_FAX_SUBMIT_FAIL	• Fax submission failed.
Includes:	genra.h	

■ Description

The **grtSubmitFax( )** function submits a queue record to a channel. This function sends a queue record to a given channel without using the queue file, to the dispatcher using the control pipe. If the submission is successful, GRT_SUCCESS is returned; otherwise, GRT_FAX_SUBMIT_FAIL is returned.

■ Example

```
#include "gfq.h"
#include "genra.h"

GFQRECORD qr;
int status, chan = 1;
int application_type = RECV_FAX_ONLY;

gfqClearRec(&qr);
switch (application_type)
{
case SEND_CALL_PROGRESS:
     /* fax channel does dialing and handles call progress */
      qr.operation = GFQDIAL_SEND;
      strcpy(qr.fn_send, "c:\\fax\\test001.tif");
      strcpy(qr.phone_no, "1-408-555-1212");
      break;
case SEND_NO_CALL_PROGRESS:
      /* fax channel goes off-hook and begins T.30 action at
         Phase B */
      qr.operation = GFQDIAL_SEND;
      strcpy(qr.fn_send, "c:\\fax\\test001.tif");
      strcpy(qr.phone_no, "\0");
      break;
case RECV_FAX_ONLY:
     /* fax channel detects incoming ring prior to going
        off-hook */
      qr.operation = GFQANSWER_RECEIVE;
      strcpy(qr.fn_received, "c:\\faxr\\f001p001.tif");
      break;
case RECV_VOICE_FAX:
     /* fax channel goes off-hook and transmits CED */
      qr.operation = GFQANSWER_IMMEDIATELY;
      strcpy(qr.fn_received, "c:\\faxr\\f001p001.tif");
      break;
}

status = grtSubmitFax(chan, &qr);
if (status == GRT_SUCCESS)
      printf("fax submitted successfully\n");
else
      printf("grtSubmitFax error %d\n", status);
```

GFD API Functions

The following section alphabetically lists the GFD API functions. These function calls handle event notification, remote status and control functions.

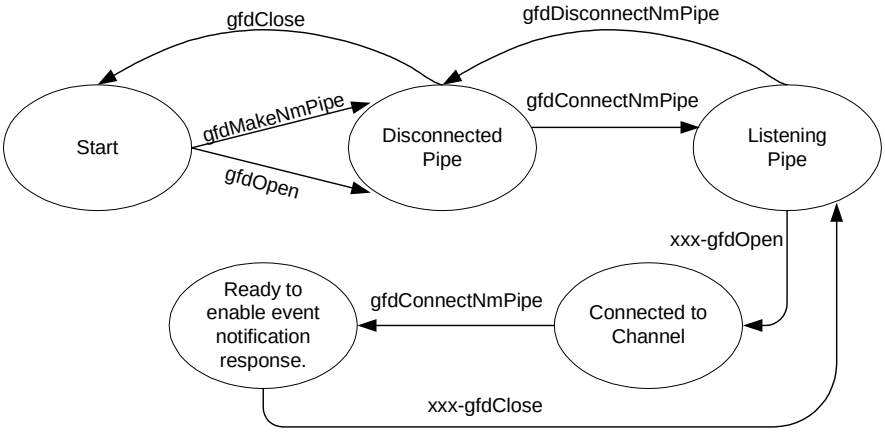

Figure 5. Event Breakpoints

Each event breakpoint may be associated to the sending side or the receiving side of a fax transmission. In the information associated with the event there may be data. The application attempts to read the gfx_rt_message, which is the largest possible message from the pipe (2048 bytes of data). There will be more or less data depending on the event. Events that are not able to respond should not be setup for response. Doing so can cause unpredictable behavior. Of the available responses, the verb response is not supported in GRT. Aborting at the Call Term break point does not abort the fax transmission at all; rather it aborts writing the queue record in the event data to the queue file on the hard disk.

Table 35. Event Breakpoints

Event	Sending/ Receiving	Information with event	Able to Respond?	Available responses
Dial	Sending	Dial string	No	None
Call Pending	Receiving	Collected digits	Yes	Abort, continue, or verb
Answer	Receiving	Collected digits	Yes	Abort, continue, queue record or verb
Info Exchange	Receiving	gfx_rt_info_s or GRT_INFO_DATA	Yes	Abort or continue
DIS Received	Sending	gfx_rt_info_s or DIS string	No	None
DCS Received	Receiving	gfx_rt_info_s or DCS string	No	None
Page Break	Sending /Receiving	None	No	None
Idle	Sending /Receiving	None	No	None
Call Term	Sending /Receiving	GFQRECORD	Yes	Abort or continue

NOTE: The GRT programming model does not support all break points and all response types covered in the GFD API. There are two additional event break points: Answer and Idle. There is also the possible response of a verb. Since this is only useful when responding to an Answer event, no additional functionality is lost.

Name:	int gfdClose (int handle)	
Inputs:	int handle	• Valid pipe handle returned from gfdMakeNmPipe or gfdOpen.
Outputs:	none	
Returns:	GFD_INVALID_HANDLE	• Wrong pipe handle.
	GFD_NOT_INSTALLED	• Dispatcher is not running.
	GFD_SUCCESS	• Successful.
Includes:	gfdmsg.h gfdipc.h	
	NOTE: gfidpc.h requires that the gfdmsg.h file be included first.	

■ Description

The **gfdClose** function closes a pipe opened with gfdOpen or gfdMakeNmPipe.

■ Example

```
#define XXX_BASE     0x4000
#define XXX_gfdClose (XXX_BASE+81)
int faxHandle; /* fax channel handle for pipe remote open */
int appHandle; /* application handle for pipe*/
int status;
int result;
/* Close remote/fax channel side of pipe */
status = gfdRemoteRequest(0, 0, XXX_gfdClose, 0, faxHandle, "", 0,
&result);
gfdDisConnectNmPipe(appHandle);
gfdClose(appHandle);
```

Name:	int gfdConnectNmPipe (int handle)	
Inputs:	int handle	• Valid pipe handle returned from gfdMakeNmPipe or gfdOpen.
Outputs:	none	
Returns:	GFD_ACCESS_DENIED	• Pipe is not connected.
	GFD_BROKEN_PIPE	• Pipe was disconnected.
	GFD_INVALID_HANDLE	• Wrong pipe handle.
	GFD_NOT_INSTALLED	• Dispatcher is not running.
	GFD_SUCCESS	• Successful.
Includes:	gfdmsg.h gfdipc.h	

■ Description

The **gfdConnectNmPipe** function closes the server handle of a named pipe. If the client end of a named pipe is open, this call forces that end of the named pipe closed. The client receives an error value on the next attempt to access the pipe.

A client that is forced off a pipe by this function must close its end of the pipe using the **gfdClose** function.

■ Example

```
#define XXX_BASE    0x400
#define XXX_gfdClose  (XXX_Base+81)

intfaxChandle;  /* Fax channel handle from remote open */
int AppHandle
int status;
int result;
/* Close remote or fax channel side of pipe */
status = gfdMakeNmPipe (*\\PIPE\\GFAX01î,ppHandle);
status = gfdConnectNmPipe(appHandle);
/* Will always return a vallue of 1 */
status = gfdRemoteRequest(1,0,XXX_gfdOpen,0,0,
     ì\\PIPE\\GFAX01î,size of(i\\PIPE\\GFAX01î),&result);
status = gfdConnectNmPipe (appHandle);
/* now will retun GFD_SUCCESS*/
```

Name:	int gfdDisConnectNmPipe (int handle)	
Inputs:	int handle	• Valid pipe handle returned from gfdMakeNmPipe or gfdOpen.
Outputs:	none	
Returns:	GFD_ACCESS_DENIED	• Pipe is not connected.
	GFD_BROKEN_PIPE	• Pipe was disconnected.
	GFD_INVALID_HANDLE	• Wrong pipe handle.
	GFD_NOT_INSTALLED	• Dispatcher is not running.
	GFD_SUCCESS	• Successful.
Includes:	gfdmsg.h gfdipc.h	

■ Description

The **gfdDisConnectNmPipe** function closes the server handle of a named pipe. If the client end of a named pipe is open, this call forces that end of the named pipe closed. The client receives an error value on the next attempt to access the pipe.

A client that is forced off a pipe by this function must close its end of the pipe using the **gfdClose** function.

■ Example

```
#define XXX_BASE    0x400
#define XXX_gfdClose  (XXX_Base+81)

int faxHandle;  /* Fax channel handle from remote open */
int AppHandle
Int status;
int result;
/* Close remote or fax channel side of pipe */
status = gfdRemoteRequest (0,0,XXX_gfdClose,0,faxHandle,îî,),
&result);
gfdDisConnectNmPipe(AppHandle);
gfdClose(AppHandle);
```

Name:	int gfdGetFileNumChannel (int chassis)	
Inputs:	int chassis	• The chassis number.
Outputs:	none	
Returns:	Number of active channels	• Successful.
	0	• The status file exists, but no channels are active.
	<0	• An error occurred.
Includes:	gfdstatu.h	

■ Description

The **gfdGetFileNumChannel** function returns the number of active channels in the chassis. A value less than zero indicates an error. A value of zero indicates that the status file exists and is valid, but that no fax channels are active.

■ Example

```
#include "gfdstatu.h"
nc = gfdGetFileNumChannel (1);
if (nc > = 0)
       printf ("Found %d channels.\n", nc);
else
       printf ("Can't open status file.\n");
```

Name:	int gfdGetMemNumChannel (int chassis)	
Inputs:	int chassis	• The chassis number.
Outputs:	none	
Returns:	Number of active channels	• Successful.
	0	• The status table exists, but no channels are active.
	<0	• An error occurred.
Includes:	gfdstatu.h	

■ Description

The **gfdGetMemNumChannel** function returns the number of active channels in the chassis. A value less than zero indicates an error. A value of zero indicates that the status table exists and is valid, but that no fax channels are active.

■ Example

```
#include "gfdstatu.h"
nc = gfdGetMemNumChannel (1);
if (nc > = 0)
       printf ("Found %d channels.\n", nc);
else
       printf ("status table not available.\n");
```

Name:	int gfdMakeNmPipe (char *name, int *handle);	
Inputs:	char *name	• Points to a null-terminated string that identifies the pipe.
	int *handle	• Points to a variable that receives the handle of the named pipe for the application.
Outputs:	int *handle	• Handle of the named pipe.
Returns:	GFD_ACCESS_DENIED	• Unable to create pipe.
	GFD_NO_PIPE_HANDLES	• Out of file handles when attempting to create a pipe.
	GFD_NOT_INSTALLED	• Dispatcher is not running.
	GFD_SUCCESS	• Successful.
Includes:	gfdmsg.h gfdipc.h	

■ Description

The **gfdMakeNmPipe** function creates a named pipe and returns a handle to the server. The handle can be used in subsequent read, write, and close operations.

■ Example

```
result = gfdMakeNmPipe ("\\PIPE\\GFAX01", &handle);
```

Name:	int gfdOpenStatusFile (int chassis, int mode)	
Inputs:	int chassis	• The chassis number.
	int mode	• The access mode of the file, which is defined in the include file fcntl.h. The mode passed must be O_RDONLY.
Outputs:	none	
Returns:		• Successful.
	Handle to the open status file	
	-1	• An error occurred.
Includes:	gfdstatu.h	

■ Description

The **gfdOpenStatusFile** function creates the name of the status file and attempts to open it. The path for the status file is obtained from the GFAX environment variable. Low-level functions, such as **gfdReadStatusFileHeader** and **gfdReadStatusFileRecord**, cannot be used unless **gfdOpenStatusFile** is called first successfully.

■ Example

```
#include "gfdstatu.h"
int chassis = 1; /* Default to one chassis*/
int status_fid;
status_fid = gfdOpenStatusFile(chassis, O_RDONLY);
if (status_fid < 0)
{
   printf ("No status file open.\n");
   exit (1);
}
```

Name:	int gfdQueryStatus (int dummy)	
Inputs:	int dummy	• Is a dummy parameter which is ignored. Any integer value is OK.
Outputs:	none	
Returns:	0	• The Dispatcher is running.
	non-zero	• The Dispatcher is not running.
Includes:	gfdmsg.h gfdipc.h	
	NOTE: gfdipc.h requires that the gfdmsg.h file be included first.	

■ Description

The **gfdQueryStatus** function checks whether or not the Dispatcher is running.

■ Example

```
if (gfdQueryStatus(0))
{
       printf ("GammaLink service not running.\n");
       exit (1);
}
```

Name:	int gfdRead (int handle, char *buffer, int size, int *bytesread)	
Inputs:	int handle	• Valid pipe handle returned from gfdMakeNmPipe or gfdOpen and connected with gfdConnectNmPipe.
	int size	• Specifies the number of bytes to be read from the pipe.
Outputs:	int *bytesread	• Points to the variable that receives the number of bytes read from the pipe.
	char *buffer	• Points to the buffer that receives the data.
Returns:	GFD_ACCESS_DENIED	• Pipe is not in correct state.
	GFD_INVALID_HANDLE	• Wrong pipe handle.
	GFD_NOT_INSTALLED	• Dispatcher is not running.
	GFD_SUCCESS	• Successful.
Includes:	gfdmsg.h gfdipc.h	

■ Description

The **gfdRead** function reads bytes of data, up to a specified number, from a pipe into a buffer. **gfdRead** may read fewer than the specified number of bytes if fewer are available. The return value is GFD_SUCCESS if the function is successful.

■ Example

```
int channel;
int result;
int appHandle;
struct gfx_rt_message datagram;
int len;
/* Main processing loop -- never ends. */
while(1)  /* Check for data on this pipe. */
{
 result = gfdRead(appHandle,(char*)&datagram,
 sizeof(datagram),&len);
   if((result==0) &&(len>0)){
   if(datagram.header.function == GFXRTBP_CALLTERM)
      printf("CallTermination on channel", " %d\n",
      datagram.header.source);
      /*Fax channel that sent message */
    }
       sleep (2);
 }
```

Name:	int gfdReadStatusFile (CPRECORD *rec, int channel, int chassis)	
Inputs:	int channel	• The logical channel number.
	int chassis	• The chassis number.
Outputs:	CPRECORD *rec	• The status record returned if the function completes successfully.
Returns:	GFD_SUCCESS	• Successful.
	GFDBAD_STATUS_FILE	• The status file is corrupt.
	GFDNO_STATUS_FILE	• The status file could not be found.
Includes:	gfdstatu.h gfdmsg.h gfdipc.h	
	NOTE: gfdipc.h requires that the gfdmsg.h file be included first.	

■ Description

The **gfdReadStatusFile** function reads one record from the status file. The status file is opened, and the record is read by calling **gfdReadStatusFileRecord**. The status file is then closed.

Example

```
#include "gfdstatu.h"
#include "gfdmsg.h"
#include "gfdipc.h"
int     channel, numchannel, chassis = 1;
CPRECORD       cprec;
for(channel=1; channel<=numchannel; ++channel)
{
  if (gfdReadStatusFile (&cprec, channel, chassis) == GFD_SUCCESS)
        printf ("channel %d, status = %021x," " name = %32x\n",
        cprec.cp_channel,cprec.cp_state, cprec.cp_name);
}
```

Name:	int gfdReadStatusFileHeader (int fid, struct cp_status_header_s header)	
Inputs:	int fid	• The file handle returned by gfdOpenStatusFile
Outputs:	struct cp_status_header_s header	• Pointer to the data structure maintained in memory that holds the status table.
Returns:	GFD_SUCCESS	• Successful.
	GFDBAD_STATUS_FILE	• The status file is corrupted.
	GFDNO_STATUS_FILE	• The status file could not be found.
Includes:	gfdstatu.h gfdmsg.h gfdipc.h	
	NOTE: gfdipc.h requires that the gfdmsg.h file be included first.	

■ Description

One field in the status-file header shows the number of fax channels active in the chassis. Also in the header file is a cp_status_header data-structure definition for a header of a status file. Table 36 lists the fields in a status-file header. A data structure for the status table is in *GFDSTATU.H*, which is the header file for status functions included in GDK. This structure is listed below:

```
struct cp_status_header_s {
       short cp_status_version; /* version of status file */
       short cp_header_size;    /* size of header (bytes) */
       short cp_record_size;    /* allocated size for each card */
       short cp_max_channel;    /* number of channels allocated */
       time_t cp_last_update;   /* time of last update */
};
```

■ Example

```
#include "gfdstatu.h"
#include "gfdmsg.h"
#include "gfdipc.h"
struct cp_status_header_s header;
/*Display header information*/
gfdReadStatusFileHeader (status_fid, &header);
printf ("Status file is version %d\n", header.cp_status_version);
printf ("Status file reports %d active",    " channels\n",
header.cp_max_channel);
printf ("Status file last updated %s\n", ctime
(&header.cp_last_update));
```

Table 36. Fields in the Status-File Header

Data Type	Field Name	Description
int	cp_status_version	Version of the status file.
int	cp_header_size	Size of the header in bytes.
int	cp_record_size	Allocated size for each fax channel.
int	cp_max_channel	Number of fax channels active in the chassis.
time_t	cp_last_update	Time of last update.

Name:	int gfdReadStatusFileRecord (int status_fid, CPRECORD *rec, int channel)	
Inputs:	int status_fid	• The file handle for the status file (returned by gfdOpenStatusFile).
	int channel	• The logical-channel number for which the status is to be obtained.
Outputs:	CPRECORD *rec	• The status record returned if the function is successful.
Returns:	GFD_SUCCESS	• Successful.
	GFDBAD_STATUS_FILE	• The status file is corrupt.
	GFDNO_STATUS_FILE	• The status file could not be found.
Includes:	gfdstatu.h gfdmsg.h gfdipc.h	
	NOTE: gfdipc.h requires that the gfdmsg.h file be included first.	

■ Description

The **gfdReadStatusFileRecord** function that reads a record from the status file if the file already is open.

Example

```
#include "gfdstatu.h"
#include "gfdmsg.h"
#include "gfdipc.h"
CPRECORD cprec;
int status_fid;
int i;
int status;
for (i = 1; i <= maxchannel; ++i)
{
       status = gfdReadStatusFileRecord    (status_fid,&cprec,i);
}
```

Name:	int gfdReadStatusMem (CPRECORD *rec, int channel, int chassis)	
Inputs:	int channel	• The logical channel number.
	int chassis	• The chassis number.
Outputs:	CPRECORD *rec	• The status record returned if the function completes successfully.
Returns:	GFD_SUCCESS	• Successful.
	GFDBAD_STATUS_FILE	• The status table is corrupt.
	GFDNO_STATUS_FILE	• The status table could not be found.
Includes:	gfdstatu.h gfdmsg.h gfdipc.h	
	NOTE: gfdipc.h requires that the gfdmsg.h file be included first.	

■ Description

The **gfdReadStatusMem** function reads one record from the status table. The status table is opened, and the record is read by calling **gfdReadStatusMemRecord**. The status table is closed without further function calls.

■ Example

```
#include "gfdstatu.h"
#include "gfdmsg.h"
#include "gfdipc.h"
int channel, numchannel, chassis = 1;
CPRECORD cprec;
numchannel = gfdGetMemNumChannel(chassis);
for(channel = 1; channel <= numchannel; ++channel)
{
       if (gfdReadStatusMem (&cprec, channel, chassis) =
GFD_SUCCESS)
         printf ("channel %d, status = %021x,", " name = %32x\n",
      cprec.cp_channel, cprec.cp_state, cprec.cp_name);
}
```

Name:	int gfdReclaimNmPipe (char *pipename)	
Inputs:	char *pipename	• Pointer to null terminated string that identifies the pipe
Outputs:	None	
Returns:	GFD_SUCCESS	• The name of the pipe has been successfully reclaimed
	GFD_INVALID_PIPE_NAME	• The name of the pipe was never used (not an error).
Includes:	gfdmsg.h gfdipc.h	
	NOTE: gfdipc.h requires that the gfdmsg.h file be included first.	

■ Description

This function performs clean up and recovers locked system resources if gfdMakeNmPipe fails with a GFD_INVALID_HANDLE error code (the typical error code that is generated from an abnormal termination of a previously running GDK fax application.)

This function can be called prior to a making gfdMakeNmPipe call without incident, but do not call this function after a successful gfdMakeNmPipe call.

■ Example

```
result = gfdReclaimNmPipe( "\\PIPE\\GFAX01" );}
```

Name:	int gfdRemoteRequest (int chassis, int channel, int command, int func, int parm, char buffer, int buffer_length, int *res)	
Inputs:	int chassis	• Chassis number (always zero).
	int channel	• Logical channel number (use 0 for the Dispatcher, GFDCP).
	int command	• A command from Table 37.
	int func	• First parameter.
	int parm	• Second parameter.
	char *buffer	• Command buffer.
	int buffer_length	• Length of command buffer.
Outputs:	int *res	• Result of the command when it is processed.
Returns:	0 or GFD_SUCCESS	• Successful. Check the result code passed back (res) for result of the actual command.
	GFDCHANNEL_FAILED	• State of the target channel is “failed.”
	GFDINVALID_CHANNEL	• Specified channel is not in the range for which the Dispatcher is configured.
	GFDREMOTE_SPARM_ERROR	• Buffer length specified in gfdRemoteRequest is greater than internal buffer size (currently, 1024).

	GFD_ACCESS_DENIED	• Failed to properly open or close the remote (fax) handle of the named pipe.

Includes: gfdmsg.h
gfdipc.h
gfqctl.h
gfq.h

■ Description

The **gfdRemoteRequest** function allows the calling application to send a message to the fax channel to perform the requested function or operation. “Remote Request” refers to the fact that the host application is requesting the “remote” software to execute certain commands or change certain parameters.

The **gfdRemoteRequestcall** contains the chassis and channel number for the communication subsystem component to execute the remote function. The channel number is the logical channel number within a chassis. Channel number 0 is used to refer to the GDK Dispatcher for that chassis. Channel number 1 is the first logical fax channel, channel number 2 is the second logical fax channel, and so forth.

There are several types of functions that the **gfdRemoteRequest** can issue. You can use **gfdRemoteRequest** to issue configuration commands from the *gfax.$dc* file. Some configuration commands have only one argument after the channel number. GFXSHUTDOWN is one such command. A typical GFXSHUTDOWN setting in the *gfax.$dc* file is:

```
GFXSHUTDOWN <channel> 3
```

To change the shutdown state of a fax channel using the **gfdRemoteRequest**, the following can be used:

```
status = gfdRemoteRequest (0, channel,GFXSHUTDOWN, 0, 3, “”, 0,
&result);
```

Other configuration commands have two arguments. GFXECM has one argument to enable ECM (error correction mode) for send mode and another argument to enable ECM for receive mode. A typical argument to enable ECM for send only is:

GFXECM <channel> 1 0

The equivalent **gfdRemoteRequest** call is:

```
status = gfdRemoteRequest(0, channel, GFXECMMODE, 1, 0, “”, 0,
&result);
```

For the configuration commands that require one argument, that argument is passed as the fifth argument to **gfdRemoteRequest**, using the function parameter. For configuration commands that require two arguments, they are passed in order as the fourth and fifth parameters to **gfdRemoteRequest**.

In the section discussing the creation of user-defined pipes and enabling event notification and response, we saw that the **gfdRemoteRequest** function is used to send messages to the fax channel. The following examples show how to use **gfdRemoteRequest** to send the most common messages relating to pipe creation, event notification, and event response.

■ Example

```
int status;
int result;

/* To open the remote (fax) end of a pipe already created using
gfdMakeNmPipe():*/
status = gfdRemoteRequest(0, channel, XXX_gfdOpen, 0, 0, pipeName,
strlen(pipename) + 1, faxHhandle):
/* To close the remote (fax) end of a pipe:*/
status = gfdRemoteRequest(0, channel, XXX_gfdClose, 0, faxHandle,
NULL, 0, &result);
/* To enable event notification only for the INFOEXCHANGE event:*/
status = gfdRemoteRequest(0, channel, GFXRTNHANDLE,
GFXRTBP_INFOEXCHANGE, faxHandle, NULL, 0, &results);
/* To enable event response on a different handle for the
INFOEXCHANGE event:*/
status = gfdRemoteRequest(0, channel, GFXRTRHANDLE,
GFXRTBP_INFOEXCHANGE, responseHandle, NULL, 0, &result);
/*To disable the default action for the CALLTERM event:*/
status = gfdRemoteRequest(0, channel, GFXRTACTION,
GFXRTBP_CALLTERM, GFXRT_ABORT, NULL, 0, &result);
/*To change the timeout for the INFOEXCHANGE event to 5 seconds:*/
status = gfdRemoteRequest(0, channel, GFXRTTIMEOUT,
GFXRTBP_INFOEXCHANGE, 5, NULL, 0, &result);
```

■ Additional Information

The **gfdRemoteRequest** also provides one more capability. When using the interactive fax programming model, the **gfdRemoteRequest** is used to submit queue records to the fax channel to give the fax channel its operation. To send a fax, for example, a queue record must be created and filled out, and then submitted to the fax channel via **gfdRemoteRequest** as in the code fragment below:

```
GFQRECORD qrec;
qrec.operation = GFQDIAL_SEND;
strcpy(qrec.fn_send, "test001.tif");
strcpy(qrec.phone_no, "1-408-744-1900");
status = gfdRemoteRequest(0, channel, GFXQRECORD, 0, 0,
(char *)&qrec, sizeof(GFQRECORD), &result);
```

To send a message to the fax channel telling it to wait for an incoming call, and then receive a fax, a similar **gfdRemoteRequest** is used:

```
GFQRECORD qrec;
qrec.operation = GFQANSWER_RECEIVE;
strcpy(qrec.fn_received, "a001p001.tif");
status = gfdRemoteRequest(0, channel, GFXQRECORD, 0, 0,
(char *)&qrec, sizeof(GFQRECORD), &result);
```

The **gfdRemoteRequest** is always used to send queue records in the same way as above. Typically, the only difference is the operation field of the queue record.

NOTE: More message options are specified in *GFQCTL.H*.

Table 37. gfdRemoteRequest Commands

Command	Configuration Commands[1]	Channel	Func	Parm	Buffer
Fax-Agent Control Messages					
GFXECMMODE	GFXECM	channel	rx	tx	null string
GFXFAXCONTROL	GFXFAXC	channel	arg	cmd	null string
GFXFINE	GFXFINE	channel	0	parm	null string
GFXFORMAT	GFXFORM	channel	0	parm	null string

Command	Configuration Commands[1]	Channel	Func	Parm	Buffer
GFXGFCONTROL	GFCCONTROL	channel	arg	cmd	null string
GFXMDMCONTROL	MODEMCTRL	channel	arg	cmd	null string
GFXSCANTIME	GFXSCANTIME	channel	0	parm	null string
GFXSHUTDOWN	GFXSHUTDOWN	channel	0	parm	null string
GFXSPEAKER	GFXSPEAKER	channel	0	parm	null string
GFXWAIT	GFXWAIT	channel	0	parm	null string
GFXCARRY_ON	GFXCARRYON	channel	0	parm	null string
GFXREJ_BURST	GFXREJBURST	channel	0	parm	null string
GFXREJ_COUNT	GFXREJCOUNT	channel	0	parm	null string
GFXREJ_PERCENT	GFXREJPERCENT	channel	0	parm	null string
GFXRTN_RETRAIN	GFXRTNRETRAIN	channel	0	parm	null string
GFXRTP_RETRAIN	GFXRTPRETRAIN	channel	0	parm	null string
Set Receive Directory					
GFXRECVPATH	GFXRECVPATH	channel	0	0	“path”
Output-Conversion-Agent Control Messages					
GFXBOTTOMMARGIN	GFXBOTTOMMARGIN	channel	0	parm	null string
GFXCHARSET	GFXCHARSET	channel	0	parm	null string
GFXEXTEND	GFXEXTEND	channel	0	parm	null string
GFXLEFTMARGIN	GFXLEFTMARGIN	channel	0	parm	null string
GFXPAGELENGTH	GFXPAGELENGTH	channel	0	parm	null string

Command	Configuration Commands[1]	Channel	Func	Parm	Buffer
GFXRIGHTMARGIN	GFXRIGHTMARGIN	channel	0	parm	null string
GFXSPACING	GFXSPACING	channel	0	parm	null string
GFXTABSTOP	GFXTABSTOP	channel	0	parm	null string
GFXTOPMARGIN	GFXTOPMARGIN	channel	0	parm	null string
Queue Record					
GFXQRECORD	—	channel	0	parm	qrec
Event Notifications					
GFRTACTION	—	channel	event id[2]	action[2]	null string
GFXENABLE	—	channel	event id[2]	notify/ response handle	null string
GFXRTNHANDLE	—	channel	event id[2]	notify handle	null string
GFXRTRHANDLE	—	channel	event id[2]	response handle	null string
GFXRTTIMEOUT	—	channel	event id[2]	seconds	null string
Named Pipe					
XXX_gfdClose[3]	—	0	0	0	pipe name
XXX_gfdOpen[3]	—	0	0	0	pipe name

[1] See the *GDK, Installation and Configuration Guide for Windows* for details on the configuration commands.

[2] See Table 38 for values.

[3] These commands are defined in xxxmsgs.h.

Table 38. Event Identifiers and Default Actions

Category	Symbolic Name
Event ID	
Call Answer	GFXRTBP_ANSWER
Call Pending	GFXRTBP_CALLPENDING
Call Termination	GFXRTBP_CALLTERM
DCS Received	GFXRTBP_DCS_RECV
Dial	GFXRTBP_DIAL
DIS Received	GFXRTBP_DIS_RECV
Information Exchange	GFXRTBP_INFOEXCHANGE
Page Break	GFXRTBP_PAGE_BREAK
Action	
Abort	GFXRT_QUEUE
Continue	GFXRT_CONTINUE
Queue	GFXRT_QUEUE

Obsolete APIs

The following functions have been removed from the 3.x releases of the GDK software.

- **gfdBoardDetect()** has been replaced by **glHWDetect().**
- **gfdGenerateConfig()** has been replaced by Control Panel configuration utility.

- **gfqReadOne()** has been replaced by **gfqFindNext().**
- **gfqReport()** has been replaced by **gfqFindFirst().**
- **gfdStartDispatcherSVC()** is redundant to the **Win32 StartService()** API.

6. Developing with PEB

PEB (Pulse Code Modulation [PCM] Expansion Bus)

The PEB (Pulse Code Modulation [PCM] Expansion Bus) was introduced by Dialogic Corporation in 1989. It is a high-speed, digital, Time-Division Multiplexer (TDM) communication bus for the signal computing environment. PEB is similar in concept to a T-1 span; information is transmitted digitally and divided into several conversations. Each conversation is referred to as a *timeslot*, and the allocation of the timeslots determines the functionality of the bus.

The following applications are possible with PEB:

- Fax
- Voice processing
- Speech recognition
- Switching
- Text-to-speech

The PEB environment consists of two primary component types: network interfaces and resource modules. Network interfaces attach to the telephone network and generally control the speed of the bus (provides “clock”). Depending on the type of network interface, PEB can support from 24 to 30 simultaneous channels. For example, T-1 supports 24 data channels and E-1 supports 30 data channels.

Resource modules perform some type of signal computing function, such as fax, by transmitting on and/or receiving data from a designated range of timeslots.

With the PEB Switching Handler Libraries, fax resources are allocated at runtime, and connected to active/live telephone calls. You can also configure fax resources as “batch processing only,” where jobs are created by the application, and submitted to the GDK Queue Management processing system. The PEB functions can be used with either the Batch Programming Model or the Interactive Programming Model.

Basics of a PEB System

A PEB system requires the utility GFSH.EXE to first program the data bus before running a PEB application. See Appendix B for more information about this utility.

Dialogic CP Fax hardware is compatible with Dialogic’s PEB network interface cards. Using the PEB APIs, the system can be a fax-only system and control call progress signaling, when the Dialogic card is configured for transparent mode. When Dialogic’s card is configured for signal insertion, as in a voice/fax system, the fax channels can be routed on the PEB bus on an as-needed fax resource.

The following list describes the functions of executable and configuration files needed for a PEB system, in addition to the base GDK software described in Chapter 2.

GFSH.CMD	Configuration file created by the timeslot assignment program and used by the GFSH.EXE program to properly assign timeslots within the SCbus system.
GFSH.EXE	The SC2000 configuration utility. It creates the GFSH.SAV file when given GFSH.CMD as input.
GFSH.SAV	Configuration file created by the GFSH.EXE program GFAX.SAV is used by the runtime API for timeslot record keeping.

PEB APIs

This section provides information about the use of the following PEB APIs:

gl_route	Connects and disconnects digital timeslot transmit and receive to PEB
gl_routerxtx	Connects and disconnects independent digital timeslot transmit and receive to PEB
gl_pebenter	Initializes PEB subsystem
gl_pebexit	Shuts down PEB subsystem

The following sections list the name, inputs, returns, include files, description, and example code for each of the PEB APIs.

Name:	int gl_route (int bddev, int chan, int tslot)	
Inputs:	int bddev	• Purpose: Unused — for compatibility with Dialogic software. • Range: Must be -1.
	int chan	• Purpose: Specifies logical fax channel. • Range: 1 – 60, inclusive.
	int tslot	• Purpose: Specifies transmit and receive PEB timeslot. • Range: 1 – 30, inclusive, for E1, 1 – 24, inclusive, for T1 to connect a timeslot to the fax channel, -1 to disconnect the timeslot of the fax channel.
Returns:	0	• Success.
	-1	• Error.
Includes:	gfpeb.h	

■ Description

The **gl_route( )** function connects and disconnects fax channel to PEB timeslot.

The gl_route function establishes or removes a connection between a logical fax channel and a PEB timeslot. The timeslot is used by the fax channel for both transmitting to and receiving from the PEB. If gl_route( ) is called to establish a connection and the fax channel is already connected to a PEB timeslot, the existing connection will be removed before the specified connection is established.

If gl_route( ) is called to remove a connection and no connection currently exists between the fax channel and a PEB timeslot, gl_route( ) will return an indication of success.

Example

```
#include "gfpeb.h"
int faxchan, status;
/* disconnect faxchan from transmit and receive timeslots on
   PEB */
status = gl_route(-1, faxchan, -1);
if (status)
{
  fprintf(stderr, "gl_route( ) disconnect failed: %d\n", status);
  return (status);
}
/* connect faxchan transmit and receive to timeslot 1 on PEB */
status = gl_route(-1, faxchan, 1);
if (status)
{
  fprintf(stderr, "gl_route( ) connect failed:" " %d\n", status);
  return (status);
}
```

Errors

- This function fails if an invalid channel is specified.
- This function fails if called on a device in SCbus mode.
- This function fails if gl_pebenter( ) has not been previously called.

Name:	int gl_routerxtx (int bddev, int chan, int rxslot, int txslot)	
Inputs:	int bddev	• Purpose: unused — for compatibility with Dialogic software. • Range: Must be -1.
	int chan	• Purpose: logical fax channel. • Range: 1 – 60, inclusive.
	int rxslot	• Purpose: receive PEB timeslot. • Range: 1 – 30, inclusive, for E1; 1 – 24, inclusive, for T1 to connect a receive timeslot to the fax channel; 0 to leave current receive timeslot configuration of fax channel unchanged; -1 to disconnect the receive timeslot of fax channel.
	int txslot	• Purpose: transmit PEB timeslot. • Range: 1 – 30, inclusive, for E1; 1 – 24, inclusive, for T1 to connect a transmit timeslot to the fax channel; 0 to leave current transmit timeslot configuration of fax channel unchanged; -1 to disconnect the transmit timeslot of fax channel.
Returns:	0	• Success.
	-1	• Error.
Includes:	gfpeb.h	

■ Description

The **gl_routerxtx ( )** function connects and disconnects fax channels to transmit and receive PEB timeslots.

The gl_routerxtx( ) function establishes a connection between a logical fax channel and a pair of PEB timeslots. The transmit and receive timeslots are selected independently. Either or both timeslots can be disconnected. Either or both timeslots can be left unchanged.

If gl_routerxtx( ) is being used to establish a connection and the fax channel is already connected to a PEB timeslot, the existing connection will be removed before the specified connection is established.

If gl_routerxtx( ) is being used to remove a connection and no connection currently exists between the fax channel and a PEB timeslot gl_routerxtx( ) will return an indication of success.

■ Example

```
#include "gfpeb.h"
int faxchan, faxchan2, txslot, rxslot, status;
/* disconnect faxchan from transmit and
       receive timeslots from PEB */
txslot = rxslot = -1;
faxchan = 1;
faxchan = 2;
status = gl_routerxtx(-1, faxchan, rxslot,
      txslot);
if (status)
{
   fprintf(stderr, "gl_routerxtx( ) disconnect" " failed: %d\n",
status);
   return (status);
}
/* connect faxchan transmit to timeslot 0 on PEB */
txslot = 0;
/* connect faxchan receive to timeslot 1 on PEB */
rxslot = 1;
status = gl_routerxtx(-1, faxchan, rxslot,
       txslot);
```

```
if (status)
{
    fprintf(stderr, "gl_routerxtx( ) connect failed: %d\n",
status);
    return (status);
}
/* To full-duplex connect faxchan with faxchan2 (e.g. for testing
purposes) perform another gl_routerxtx( ) for faxchan2 with the
txslot and rxslot parameters reversed. */
```

■ Errors

- This function fails if an invalid channel is specified.
- This function fails if called on a device in SCbus mode.
- This function fails if gl_pebenter( ) has not been previously called.

Name:	int gl_pebenter (void)	
Inputs:	none	• A pointer to a queue record.
Returns:	0	• Success.
	-1	• Error.
Includes:	gfpeb.h	

■ Description

The **gl_pebenter( )** function initialize PEB subsystem. The gl_pebenter( ) function must be called before using any PEB switching functions.

■ Example

```
#include "gfpeb.h"
int status;
status = gl_pebenter( );
if (status)
{
   fprintf(stderr, "gl_pebenter( ) failed: %d\n", status);
   return (status);
}
```

■ Errors

- This function fails if the GFSH.SAV file cannot be found and opened with read and write privileges.
- This function also fails if %GFAX% is not defined.

Name:	int gl_pebexit ( )	
Inputs:	none	
Returns:	0	• Success.
	-1	• Error.
Includes:	gfpeb.h	

■ Description

The **gl_pebexit ( )** function shuts down PEB subsystem. The gl_pebexit( ) function must be called before program termination.

■ Example

```
#include "gfpeb.h"
int status;
status = gl_pebexit( );
if (status)
{
   fprintf(stderr, "gl_pebexit( ) failed: %d\n", status);
   return (status);
}
```

7. Developing with SCbus

SCbus Connectivity Paradigm

SCbus (Signal Computing) bus is an open architectural specification for digital intra-node communication. SCbus is a high-capacity bus with up to 2,048 time-slots, which will handle most audio/video applications. By using the SCbus connectivity paradigm, you can focus on other application issues (such as resource allocation) instead of complex connectivity problems. Each SCbus-compliant product provides virtually the same software interface to its given resource(s). Another advantage of developing with SCbus is consistency among compliant products.

While describing the SCbus characteristics in general, this section covers GDK SCbus-specific topics and introduces the SCbus Application Program Interfaces (APIs) for the CP Fax SC boards. Because of SCbus requirements, each compliant product must assign each of its resources to an unchanging and permanent transmit timeslot on the bus. The procedures to “nail up” transmit timeslots for its fax channels are described in detail at the end of this section.

Basics of SCbus Compliancy

Dialogic CP Fax software and programs must interact with an existing, properly installed Dialogic SCbus system. This is necessary to coordinate timeslot assignment issues.

The following list describes the functions of the executable and configuration files:

GFSH.BAS	Configuration file created by the Configure SCbus utility. This file indicates the number of CP Fax SC boards in the chassis.
GFTSASGN.EXE	GDK's timeslot assignment program. This program is called by Dialogic's master timeslot assignment program during the Dialogic system startup. GFTSASGN creates GFSH.CMD, given the total number of consecutive timeslots assigned to the GDK system and the starting timeslot number.
GFSH.CMD	Configuration file created by the timeslot assignment program and used by the GFSH.EXE program to properly assign timeslots within the CP Fax SCbus system.
GFSH.EXE	The SC2000 configuration utility. It creates the GFSH.SAV file when given GFSH.CMD as input.
GFSH.SAV	Configuration file created by the GFSH.EXE program GFAX.SAV is used by the runtime API for timeslot record keeping.
GFTSREQ.DAT	This file is created by the Configure SCbus utility. It contains the timeslot requirements and the name of the assignment program (GFTSASGN.EXE).

SCbus APIs for the CP Fax SC Boards

GDK supports all SCbus APIs for the CP Fax SC boards as follows:

gl_getctinfo	Returns SCbus device information
gl_getxmitslot	Returns specified fax channel's transmit timeslot on the SCbus
gl_listen	Connects the fax receive channel to the SCbus timeslot

gl_unlisten	Disconnects the fax receive channel from the SCbus timeslot
gl_scenter	Initializes the SCbus subsystem
gl_scexit	Shuts down the SCbus subsystem

ScBus APIs With DM3 Boards

Using SCBus mode with DM3 boards requires special function calls (_Ex functions). The CP SCBus API bypasses the dispatcher by accessing a file and communicating directly with the driver. These _Ex functions "remap" the existing CP/ScBus API function calls to redirect the SC Bus back to the Dispatcher.

GDK developers must recomplile their applications after using the _Ex functions.

NOTE: You must allocate the ScBus timeslot ***prior*** to starting a fax session. Perform these steps:

1. Call the gl_listen function prior to submitting a queue record (Qrec) using GFQANSWER_IMMEDIATELY or GFQDIAL_SEND. This eliminates error code 3830 (GX0_NODIAL or no dial tone detected prior to dialing).

2. Use the gl_unlisten function call to free the ScBus timeslot when the line is disconnected. Please note that gl_listen and gl_unlisten will always return a –1 (not in the sample provided, but in the original file included in the install).

The _Ex functions are available under \fax\gl_scbus.cpp. These five macros should be defined in your header file before they can be remapped to the appropriate function:

```
#define gl_getxmitslot gl_getxmitslotEx
#define gl_listen gl_listenEx
#define gl_unlisten gl_unlistenEx
#define gl_scenter gl_scenterEx
#define gl_scexit gl_scexitEx
```

The following is an example of a header file defining the _Ex functions:

```
//      Gammalink Header files

#ifdef __cplusplus

extern "C" {

#endif

#include "gamma.h"

#include "gfdboard.h"

#include "gfq.h"

#include "gfqpath.h"

#include "genra.h"

#include "gfxevent.h"

#include "gfqctl.h"

#include "gfdmsg.h"

#include "gfdipc.h" //       <--- contain gfdRemoteFunction
definition

#include "gfdstatu.h"

#include "gfxstate.h"

#include "gfsc.h"  // <--- contain the original gl_listen,...

#ifdef __cplusplus

};

#endif
```

The following is the gl_scbus.ccp source code:

```
#define GFDLISTEN            GFDBASE+100
```

```
#define GFDUNLISTEN          GFDBASE+101

#define GFDGETTIMESLOT       GFDBASE+102

#define GFDSCENTER           GFDBASE+103

#define GFDSCEXIT            GFDBASE+104

#define gl_getxmitslot gl_getxmitslotEx

#define gl_listen gl_listenEx

#define gl_unlisten gl_unlistenEx

#define gl_scenter gl_scenterEx

#define gl_scexit gl_scexitEx


int gl_getxmitslotEx( int chan, SC_TSINFO* tsinfop )

{

       int    status,result;

       status=gfdRemoteRequest(0,chan,GFDGETTIMESLOT,0,0,NULL,0,&r
esult);

       if (status)

       {

              tsinfop->sc_tsarrayp[0] = 0;

              return -1;

       } else {

              tsinfop->sc_tsarrayp[0] = result;

              return 0;

       }
```

```
}


int gl_listenEx( int chan, SC_TSINFO* tsinfop )

{

        int    status,result;

        int timeslot = tsinfop->sc_tsarrayp[0];

        status=gfdRemoteRequest(0,chan,GFDLISTEN,
timeslot,timeslot,NULL,0,&result);

        if (status)

        {

                return -1;

        } else {

                return 0;

        }

}


int gl_unlistenEx(int chan)

{

        int    status,result;

        status=gfdRemoteRequest(0,chan,GFDUNLISTEN,
0,0,NULL,0,&result);

        if (status)

        {

                return -1;
```

```
        } else {

                return 0;

        }

}



int gl_scenterEx(void)

{

        int     status,result;

        status=gfdRemoteRequest(0,0,GFDSCENTER, 0,
0,NULL,0,&result);

        if (status || result)

        {

                return -1;

        }

        return 0;

}



int gl_scexitEx(void)

{

        int     status,result;

        status=gfdRemoteRequest(0,0,GFDSCEXIT, 0,
0,NULL,0,&result);

        if (status || result)

        {
```

```
            return -1;

        }

        return 0;

}
```

Scbus API Descriptions

The following sections list the name, inputs, outputs, returns, include files, description, and example code for each of the SCbus APIs.

Name:	int gl_getctinfo (int chan, CT_DEVINFO *ct_devinfop)	
Inputs:	int chan	• Purpose: CP fax channel. • Range: 1 - n, where n is the number of installed channels.
	CT_DEVINFO *ct_devinfop	• Purpose: Pointer to SCbus timeslots information structure. • Range: N/A.
Returns:	0	• Success.
	-1	• Error.
Includes:	gfsc.h	

■ Description

The **gl_getctinfo( )** function returns digital timeslot device information.

This function retrieves the device information related to a channel on a CP Fax SC board. On return from the function, the CT_DEVINFO structure contains the relevant information. The CT_DEVINFO structure is declared as follows:

```
typedef struct {
unsigned longct_prodid;
unsigned charct_devfamily
unsigned charct_devmode;
unsigned charct_nettype;
unsigned charct_busmode;
unsigned charct_busencoding;
unsigned charct_rfu[7];
} CT_DEVINFO;
```

The valid values for each member of the CT_DEVINFO structure are defined in *gfsc.h*.

The ct_prodid field contains a valid CP Fax product identification number for the device. The ct_devfamily field specifies the device family and will contain one of the following values:

CT_DFCP4 /* GammaLink CP -4 family */

CT_DFCP6 /* GammaLink CP -6 family */

CT_DFCP12 /* GammaLink CP -12 family */

The ct_devmode field is not valid for the CP Fax Series card.

The ct_nettype field may contain either of the following values:

CT_NTT1 /* T1 configuration */

CT_NTE1 /* E1 configuration */

CT_NTNONE

The ct_busmode field may contain either of the following values:

CT_BMPEB /* PCM Expansion Bus architecture */

CT_BMSCBUS /* SCbus architecture */

The ct_busencoding field may contain either of the following values:

CT_BEULAW /*Mu-law PCM encoding */

CT_BEALAW /*A-law PCM encoding */

■ Example

```
#include "gfsc.h"
CT_DEVINFO devinfop;
      .
      .
      .
status = gl_getctinfo(gl_devh, &devinfop)
if(status == 0)
   printf("Fax channel %d: product id=%d, bus mode=%d\n",
          gl_devh, devinfop.ct_prodid, devinfop.ct_busmode);
else
   printf("Error %d\n", status);
```

Name:	int gl_getxmitslot (int chan, SC_TSINFO *tsinfop)	
Inputs:	int chan	• Purpose: GDK channel. • Range: 1 – n, where n is the number of installed fax channels.
	SC_TSINFO * tsinfop	• Purpose: Pointer to SCbus timeslots information. • Range: N/A.
Returns:	0	• Success.
	-1	• Error.
Includes:	gfsc.h	

■ Description

The **gl_getxmitslot( )** function returns SCbus timeslot connected to digital timeslot transmit.

This function connects the external SCbus timeslot to the transmit of a channel on a CP Fax SC board. On return from the function, the SC_TSINFO structure contains the number of SCbus timeslots that are connected to the transmit of the local fax timeslot and a pointer to the array that contains the SCbus timeslots (between 1 and 1024). The SC_TSINFO structure is declared as follows:

```
typedef struct {
        unsigned longnumts;
        long*tsarray;
} SC_TSINFO;
```

This function fails if an invalid channel is specified. This function fails if called on a device in PEB mode.

NOTE: A CP Fax SC fax channel can only transmit on one external SCbus timeslot. The SC_TSINFO structure is used to provide uniformity among SCbus timeslot access functions for all SCSA devices.

Example

```
#include "gfsc.h"
SC_TSINFO   sc_tsinfo;   /* SCbus timeslot info
structure */
long  scts; /* SCbus timeslot */
int   devh1, status;

devh1 = 1;
sc_tsinfo.sc_numts = 1; /* always one timeslot */
sc_tsinfo.sc_tsarrayp = &scts;
status = gl_getxmitslot( devh1, &sc_tsinfo );
if (status)
{
   fprintf(stderr, "gl_getxmitslot( ) error %d\n", status);
   return (status);
}
else
   fprintf(stdout,"chan %d is transmitting on timeslot %d\n",
           devh1,scts);
```

Name:	int gl_listen (int chan, SC_TSINFOP *tsinfop)	
Inputs:	int chan	• Purpose: CP Fax SC channel. • Range: 1 – n, where n is the number of installed fax channels.
	SC_TSINFO * tsinfop	• Purpose: Pointer to SCbus timeslots information structure. • Range: N/A.
Outputs:	GFQRECORD *qrec	• A pointer to a queue record.
Returns:	0	• Success.
	-1	• Error.
Includes:	gfsc.h	

■ Description

The **gl_listen( )** function connects the fax channel to receive timeslot on SCbus

This function connects the receive data stream of a fax channel to a timeslot on the external SCbus. The SC_TSINFO structure contains two fields. The first field specifies the total number of SCbus timeslots to connect. The second field is a pointer to an array that contains the SCbus timeslots (from 1 and 1024) necessary to connect the receive data stream of the channel. If gl_listen( ) is called to establish a connection for a CP Fax SC fax channel that is already connected, the existing connection will be broken before the specified connection is established.

The SC_TSINFO structure is declared as follows:

```
typedef struct {
      unsigned longnumts;
      long*tsarray;
} SC_TSINFO;
```

Multiple fax channels may listen to a single SCbus timeslot, although the transmit data stream of a single SC channel can be connected to only one SCbus timeslot. The SC_TSINFO structure is used to provide uniformity among SCbus timeslot access functions for all SCSA devices.

■ Example

```
#include "gfsc.h"
SC_TSINFO   sc_tsinfo;  /* SCbus timeslot info structure */
long scts;     /* SCbus timeslot */
int  devh1, devh2, status;

devh1 = 1;
devh2 = 2;
sc_tsinfo.sc_numts = 1;   /* always one timeslot */
sc_tsinfo.sc_tsarrayp = &scts;
status = gl_getxmitslot( devh1, &sc_tsinfo );
if (status)
{
   fprintf(stderr, "gl_getxmitslot( ) error %d\n", status);
   return (status);
}
else
     fprintf(stdout, "chan %d is transmitting on timeslot %d\n",
             devh1, scts);
/* connect devh2 rxslot to devh1 txslot */
status = gl_listen( devh2, &sc_tsinfo );
if (status)
{
   fprintf(stderr, "gl_listen( ) error %d\n", status);
   return (status);
}
else
   fprintf(stdout, "chan %d is receiving on timeslot %d\n",
           devh2, scts);
/* devh2 is connected half-duplex with devh1.  For a full-duplex
connection, repeat the above steps to connect devh1 rxslot to devh2
txslot */
```

■ Error

- This function fails if an invalid channel is specified.
- This function fails if the SCbus timeslot number is invalid.
- This function fails if called on a device in PEB mode.

Name;	int gl_unlisten (int chan)	
Inputs:	int chan	• Purpose: Specifies CP Fax SC board channel.
Returns:	0	• Success.
	-1	• Error.
Includes:	gfsc.h	

■ Description

The **gl_unlisten( )** function disconnects fax channel receive data stream from SCbus. This function disconnects the receive data stream of a fax channel from the SCbus. If gl_unlisten( ) is called to disconnect a fax channel that has no existing connection, no operation will be performed and the function will return an indication of success.

■ Example

```
#include "gfsc.h"
int devh1, status;

devh1 = 1;
/* disconnect devh1 rxslot */
status = gl_unlisten( devh1 );
if (status)
{
   fprintf(stderr, "gl_unlisten( ) error %d\n", status);
   return (status);
}
```

■ Error

- This function fails if an invalid channel is specified.
- This function fails if called on a device in PEB mode.

Name:	int gl_scenter (void)	
Inputs:	None	
Returns:	0	• Success.
	-1	• Error.
Includes:	gfsc.h	

■ Description

The **gl_scenter( )** function must be called once before using any SCbus switching functions.

■ Example

```
#include "gfsc.h"
/* initialize SCbus system */
status = gl_scenter( );
if (status)
{
   fprintf(stderr, "gl_scenter( ) error %d\n", status);
   return (status);
}
```

■ Error

- This function fails if the GFSH.SAV file cannot be found and opened with read and write privileges.
- This function also fails if %GFAX% is not defined.

Name:	int gl_scexit ( )	
Inputs:	None	
Returns:	0	• Success.
	-1	• Error.
Includes:	gfsc.h	

■ Description

The **gl_scexit( )** function must be called before program termination.

■ Example

```
#include "gfsc.h"
status = gl_scexit( );
if ( status != 0 )
{
     fprintf(stderr,"failed gl_scexit:"
" %d\n",status );
}
return (status);
```

■ Error

- This function fails if the gfsh.sav file cannot be closed.

8. Fax Status Files

Overview

This chapter discusses status tables and status files. It covers these main topics:

- Status tables and status files
- Creating a status file
- Refreshing the status file
- Monitoring status with gfxStatus
- Monitoring status with cp_state
- System information API function calls

Status Tables and Status Files

The Dispatcher maintains in memory a *status table* that records the hardware setup of a specific fax channel and the processing status of fax files directed to that channel. Using the information on the status table, a *status file* can be created on disk that checks the state of the fax channels in a multiple-channel chassis or in multiple-chassis systems. Real-time status information can be obtained, and applications capable of response in real time can be built, using the status information.

Each chassis has only one status table and only one status file. Each fax channel in a chassis is represented by only one record in the status table or status file. When the status table is used on the chassis in which the boards are installed, the table should be used with the function gfdReadStatusMem for faster operation. The status table in memory and the status file on disk are a collection of records, one for each fax channel. That is, a chassis with four fax channels will have four records.

The format of a status record is listed in Table 39.

Table 39. Status Record Fields

Field Name	Data Type	Description
cp_channel	int	The physical-channel number.
cp_name [32]	char	The modem_id of a fax channel.
cp_list	int	The internal-buffer list number for transactions addressed to this fax channel.
cp_state	int	Indicates the state of the fax channel. The states include “idle,” “sending,” “receive,” “offline,” “dead,” “reset,” and “failed.”
cp_pass	int	Not used; reserved for future use.
capabilities	unsigned	Not used; reserved for future use.
ProgramFile [64]	char	Not used; reserved for future use.
ProgramOptions	unsigned	Not used; reserved for future use.
numOpenFiles	int	The number of files a fax channel has open.
cp_fid [8]	int	The system-file numbers for the files the fax channel has open. Unopened files have a 0 or -1 here.
LastFileName [64]	char	The filename of the last file that was successfully opened.
gfxStatus	int	The last gfxStatus returned by the fax channel.
gfxState	int	The last gfxState returned by the fax channel. See values in gfxstate.h.
country	int	The country code for the fax channel. (This information is requested on startup.)
gfxDebug	int	Not used; reserved for future use.

Field Name	Data Type	Description
ActiveHandle	int	Contains the queue record handle for the transaction currently on the fax channel.
ActiveQueueId	long	Not used; reserved for future use.
LastUserId[32]	char	The user_id field for the transaction currently on the fax channel.
numSend	long	Counts the number of outgoing transactions the fax channel has processed since the Dispatcher was last restarted.
TotalSend	long	The number of outgoing transactions the fax channel has processed since the status file was last reset. This field is restored from the status file when status-file processing is enabled.
numReceive	long	Counts the number of incoming transactions the fax channel has processed since the Dispatcher was last restarted.
TotalReceive	long	Counts the number of incoming transactions the fax channel has processed since the status file was last reset. This field is restored from the status file when status-file processing is enabled.
numFailed	long	Counts the number of failed transactions the fax channel has processed since the Dispatcher was last restarted. Failed transactions are also counted in numSend or numReceive as appropriate.

Field Name	Data Type	Description
TotalFailed	long	Counts the number of failed transactions that the fax channel has processed since the log file was last reset. This value is restored from the status file when status-file processing is enabled. Failed transactions also are counted in TotalSend or TotalReceive, as appropriate.
ItemsSent	long	Counts the number of items (pages) sent since the Dispatcher was last restarted.
itemsReceived	long	Counts the number of items (pages) received since the Dispatcher was last restarted.
ConnectSeconds	long	Counts the number of seconds the phone line has been off hook since the on-board software was last restarted.
LastSpeed	int	Records the speed at which the last transaction took place. This field may be useful for locating bad phone lines.
LastError	int	Not used; reserved for future use.

Creating a Status File

The status file should be used on remote workstations connected by a LAN. The name of the status file is "GFAXn.$DS," where the "n" indicates the chassis number defined in the chassis command and "DS" stands for Dispatcher status. The status file is created by the STATUST.

Refreshing the Status File

Status information is not **appended** to the status file; it is **overwritten**. When the status records are updated in the status file, the record for a particular fax channel is overwritten with the latest status information for that channel.

The STATUST command parameter specifies the interval (in seconds and milliseconds) between the posting of status records to disk. The range for the parameter value is between 0 and 32,767, with 0 being disabling; a reasonable number is 5.

NOTE: The STATUST command is for an entire chassis, and not just for one fax channel in the chassis.

Table 40. Status-Table Functions

High-Level Functions*	**Purpose**
gfdReadStatusFile	To read one status record and close the status file.
gfdGetFileNumChannel	To get the number of channels and close the status file.
gfdReadStatusMem	To read a record from the status table in memory.
gfdGetMemNumChannel	To get the number of channels by reading the status table in memory.
Low-Level Functions**	**Purpose**
gfdOpenStatusFile	To open the shared status file.
gfdReadStatusFileHeader	To read a header from an opened status file.
gfdReadStatusFileRecord	To read a record from an opened status file.

* High-level functions close the status file after processing.

** Low-level functions do not close the status file after processing.

Monitoring Status with gfxStatus

An important element in the status-table data structure is the variable gfxStatus. Bits in this field are shown in *Figure 6. Bits in the gfxStatus Field*. The first four bits reveal the fax channel shutdown states, which are an extension of the shutdown status.

The term “shutdown” describes some options that are set with the GFXSHUTDOWN command. For example, the shutdown option 0 means that a fax channel polls the Queue File Pending List for outgoing jobs and answers incoming calls. GFXSHUTDOWN 2 tells the channel to stop transmitting, although it still can receive. That is a useful arrangement in a multi-fax channel chassis in which some fax channels are dedicated to transmitting and others to receiving. An operator who wished to confirm that a channel is only receiving, and not sending, would expect bit 1 of gfxStatus to be turned on. Bits 4 to 9 of gfxStatus are the *capabilities flags*, which list the receive and transmission options that have been enabled.

The last six bits of gfxStatus are read-only. By monitoring the gfxStatus bits within the status table in memory, an application can show when each channel is receiving, sending, on-line, off-hook, or ringing.

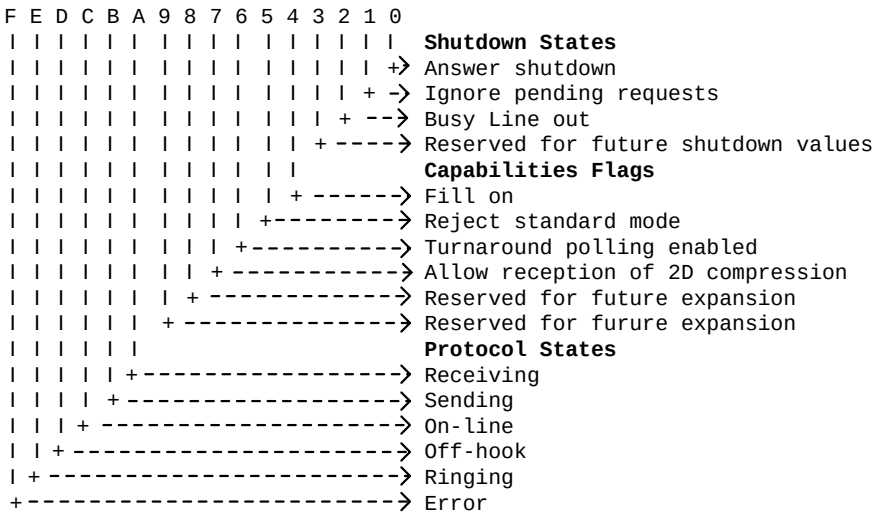

Figure 6. Bits in the gfxStatus Field

Monitoring Status with cp_state

The diagram in Figure 7 shows the various cp_states of a fax channel. “INIT” is the command used to reset and recognize a fax channel, and “LOAD” is the command issued during firmware download to the fax channel. Only INIT or LOAD can be performed on a channel where cp_ state has a value less than CP_IDLE.

The cp_state of a fax channel is listed in Table 41. These values are defined in GFDSTATU.H.

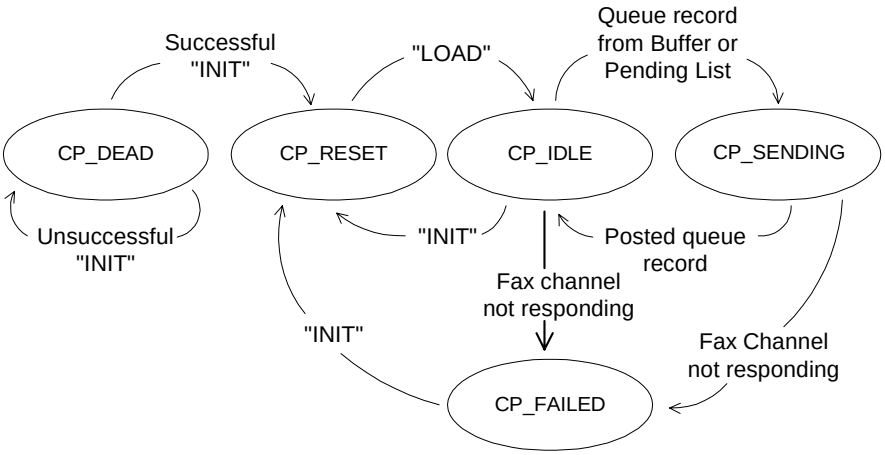

Figure 7. CP_ States

The following table defines the CP_ states.

Table 41. CP_States

State	Explanation
CP_DEAD	The fax channel has not yet been initialized successfully.
CP_RESET	The fax channel has been successfully initialized.
CP_IDLE	The on-board software has been successfully loaded, but no queue record is currently active on the fax channel.
CP_SENDING	A queue record from the Pending List is active on the fax channel.
CP_FAILED	The fax channel is not longer responding.

GDK System Information API Function Calls

The following APIs provide information about the CP product hardware and Dispatcher status. The alphabetized list of these functions follows.

Name:	int gfdGetFileNumChannel (int chassis)	
Inputs:	int chassis	• The chassis number.
Outputs:	None	
Returns:	Number of active channels	• The function was successful.
	0	• The status file exists, but no channels are active.
	<0	• An error occurred.
Includes:	gfdstatu.h	

■ Description

The **gfdGetFileNumChannel( )** function gets the number of channels from the status file. This is a high-level status function will return the number of active channels in the chassis. A value less than zero indicates an error. A value of zero indicates that the status file exists and is valid, but that no fax channels are active.

■ Example

```
#include "gfdstatu.h"
nc = gfdGetFileNumChannel (1);
if (nc > = 0)
   printf ("Found %d channels\n", nc);
else
   printf ("can't open status file\n");
```

Name:	int gfdGetMemNumChannel (int chassis)	
Inputs:	int chassis	• The chassis number.
Outputs:	None	
Returns:	Number of active channels	• The function was successful.
	0	• The status file exists, but no channels are active.
	<0	• An error occurred.
Includes:	gfdstatu.h	

■ Description

The **gfdGetMemNumChannel( )** function gets the number of channels from the status table. This is a high-level status function will return the number of active channels in the chassis. A value less than zero indicates an error. A value of zero indicates that the status table exists and is valid, but that no fax channels are active.

■ Example

```
#include "gfdstatu.h"
nc = gfdGetMemNumChannel (1);
if (nc > = 0)
   printf ("Found %d channels\n", nc);
else
   printf ("status table not available\n");
```

Name:	int gfdOpenStatusFile (int chassis, int mode)	
Inputs:	int chassis	• The chassis number.
	int mode	• The access mode of the file, which is defined in the include file fcntl.h. The mode passed must be O_RDONLY.
Outputs:	None	
Returns:	Handle to the open status file	• Successful.
	-1	• An error occurred.
Includes:	gfdstatu.h	

■ Description

The **gfdOpenStatusFile( )** function prepares a status file for reading by other functions. This routine creates the name of the status file and attempts to open it. The path for the status file is obtained from the GFAX environment variable. Low-level functions, such as gfdReadStatusFileHeader and gfdReadStatusFileRecord, cannot be used unless gfdOpenStatusFile is called first successfully.

■ Example

```
#include "gfdstatu.h"
int chassis = 1; /* Default to one chassis*/
int status_fid;
status_fid = gfdOpenStatusFile(chassis,O_RDONLY);
if (status_fid < 0)
{
   printf ("No status file open\n");
   exit (1);
}
```

Name:	int gfdQueryStatus (int dummy)	
Inputs:	int dummy	• Dummy parameter - ignored. Any integer value is OK.
Outputs:	None	
Returns:	0	• Dispatcher is running.
	non-zero	• Dispatcher is not running.
Includes:	gfdmsg.h gfdipc.h	

■ Description

The **gfdQueryStatus( )** function checks if the Dispatcher is running. This function checks whether or not the Dispatcher is running.

■ Example

```
if (gfdQueryStatus(0))
{
   printf ("GammaLink service not running\n");
   exit (1);
}
```

Name:	int gfdReadStatusFile (CPRECORD *rec, int channel, int chassis)	
Inputs:	int channel	• The logical channel number.
	int chassis	• The chassis number.
Outputs:	CPRECORD *rec	• The status record returned if the function completes successfully.
Returns:	GFD_SUCCESS	• The status record was successfully read.
	GFDBAD_STATUS_FILE	• The status file is corrupt.
	GFDNO_STATUS_FILE	• The status file could not be found.
Includes:	gfdstatu.h gfdmsg.h gfdipc.h	• gfdipc.h requires that the file gfdmsg.h be included first.

■ Description

The **gfdReadStatusFile( )** function reads one record from the status file. This high-level status function reads one record from the status file. The status file is opened, and the record is read by calling gfdReadStatusFileRecord. The status file is then closed.

Example

```
#include "gfdstatu.h"
#include "gfdmsg.h"
#include "gfdipc.h"
int   channel, numchannel, chassis = 1;
CPRECORD    cprec;
for(channel=1; channel<=numchannel; ++channel)
{
  if (gfdReadStatusFile (&cprec, channel,
      chassis) == GFD_SUCCESS)
      printf ("channel %d, status = %021x, name = %32x\n",
            cprec.cp_channel,cprec.cp_state, cprec.cp_name);
}
```

Name:	int gfdReadStatusFileHeader (int fid, struct cp_status_header_s header)	
Inputs:	int fid	• The file handle returned by gfdOpenStatusFile.
Outputs:	struct cp_status_header_s header	• Pointer to the data structure maintained in memory that holds the status table.
Returns:	GFD_SUCCESS	• The status record was successfully read.
	GFDBAD_STATUS_FILE	• The status file is corrupted.
	GFDNO_STATUS_FILE	• The status file could not be found.
Includes:	gfdstatu.h gfdmsg.h gfdipc.h	• gfdipc.h requires that the file gfdmsg.h be included first.

■ Description

The **gfdReadStatusFileHeader( )** function reads version, size, and date information from a status file.

One field in the status-file header shows the number of fax channels active in the chassis. Also in the header file is a cp_status_header data-structure definition for a header of a status file. Table 42 lists the fields in a status-file header. A data structure for the status table is in GFDSTATU.H, which is the header file for status functions included in GDK. This structure follows:

```
struct cp_status_header_s {
      short cp_status_version; /* version of status file */
      short cp_header_size;   /* size of header (bytes) */
      short cp_record_size;   /* allocated size for each card */
      short cp_max_channel;   /* number of channels allocated */
      time_t cp_last_update;  /* time of last update */
};
```

Table 42. Fields in the Status-File Header

Data Type	Field Name	Description
int	cp_status_version	Version of the status file.
int	cp_header_size	Size of the header in bytes.
int	cp_record_size	Allocated size for each fax channel.
int	cp_max_channel	Number of fax channels active in the chassis.
time_t	cp_last_update	Time of last update.

■ Example

```
#include "gfdstatu.h"
#include "gfdmsg.h"
#include "gfdipc.h"
struct cp_status_header_s header;
/*Display header inforrmation*/
gfdReadStatusFileHeader (status_fid, &header);
printf ("Status file is version %d\n", header.cp_status_version);
printf ("Status file reports %d active channels\n",
        header.cp_max_channel);
printf ("Status file last updated %s\n",
        ctime(&header.cp_last_update));
```

Name:	int gfdReadStatusFileRecord (int status_fid, CPRECORD *rec, int channel)	
Inputs:	int status_fid	• The file handle for the status file (returned by gfdOpenStatusFile).
	int channel	• The status for the logical-channel number.
Outputs:	CPRECORD *rec	• The status record returned if the function is successful.
Returns:	GFD_SUCCESS	• The function executed.
	GFDBAD_STATUS_FILE	• The status file is corrupt.
	GFDNO_STATUS_FILE	• The status file could not be found.
Includes:	gfdstatu.h gfdmsg.h	
	gfdipc.h	• gfdipc.h requires that the file gfdmsg.h be included first.

■ Description

The **gfdReadStatusFileRecord( )** function reads a record from the status file. This is a low-level status function that reads a record from the status file if the file already is open.

■ Example

```
#include "gfdstatu.h"
#include "gfdmsg.h"
#include "gfdipc.h"

CPRECORD cprec;
int status_fid;
int i;
int status;
```

```
for (i = 1; i <= maxchannel; ++i)
{
    status = gfdReadStatusFileRecord
    (status_fid,&cprec,i);
 }
```

Name:	int gfdReadStatusMem (CPRECORD *rec, int channel, int chassis)	
Inputs:	int channel	• The logical channel number.
	int chassis	• The chassis number.
Outputs:	CPRECORD *rec	• The status record returned if the function completes successfully.
Returns:	GFD_SUCCESS	• The function executed.
	GFDBAD_STATUS_FILE	• The status table is corrupt.
	GFDNO_STATUS_FILE	• The status table could not be found.
Includes:	gfdstatu.h gfdmsg.h gfdipc.h	• gfdipc.h requires that the file gfdmsg.h be included first.

■ Description

The **gfdReadStatusMem( )** function reads a record from the status table. This high-level status function reads one record from the status table. The status table is opened, and the record is read by calling gfdReadStatusMemRecord. The status table is closed without further function calls.

■ Example

```
#include "gfdstatu.h"
#include "gfdmsg.h"
#include "gfdipc.h"
int channel, numchannel, chassis = 1;
CPRECORD cprec;
numchannel = gfdGetMemNumChannel(chassis);
```

```
for(channel=1; channel<=numchannel; ++channel)
{
   if (gfdReadStatusMem (&cprec, channel, chassis) = GFD_SUCCESS)
       printf ("channel %d, status = %021x, name = %32x\n",
               cprec.cp_channel, cprec.cp_state, cprec.cp_name);
}
```

Name:	int glHWDetect (glSystem *glSys, glBoardSystem *glBdSys, int reserve2)	
Inputs:	glSystem *glSys	• Pointer to glSystem structure or NULL if this information is not desired.
	glBoardSystem *glBdSys	• Pointer to glBoardSystem structure or NULL if this information is not desired.
	int reserve2	• Must be 0 (zero).
Outputs:	glSystem *glSys	• Contains channel information if the return value is GL_HWCONFIG_SUCCESS and the pointer to glSystem was not NULL.
	glBoardSystem *glBdSys	• Contains CP Fax board information if the return value is GL_HWCONFIG_SUCCESS and the pointer to glBoardSystem was not NULL.
Returns:	GL_HWCONFIG_SUCCESS	• Successful.

GL_HWDETECT_IN_PROGRESS	• An instance of glHWDetect is already running. Wait until the first instance completes.
GL_SERVICE_RUNNING	• The GDK System Service is running. Stop the service then run glHWDetect again.
GL_SYSTEM_NULL_PTR	• Invalid parameters. The first and second parameters cannot both be NULL.
GL_SYSTEM_CHANNEL_ OVERLAP	• glHWDetect found overlapping fax channels for boardtypes CP4/SC, CP6/SC, or CP12/SC. Check that all channels in the system have unique I/O addresses, reconfigure, and run glHWDetect again.

GL_HWDRIVER_FAIL	• The required device drivers failed to start. Try starting the GDK ISA Device Driver and the GDK PCI Device Driver using the Devices Control Panel Applet. Call Technical Support for additional assistance.
GL_HWDETECT_FAIL	• Internal failure to create a system controlled resource. Fatal error, reboot the system and try again.
GL_SYSTEM_MALLOCFAILED	• Internal failure to allocate a system controlled resource. Fatal error, reboot the system and try again.

Includes: gfdboard.h

Description

The **glHWDetect** function provides information about the CP Fax hardware installed in the system. This function blocks while checking a limited area of I/O space for the presence of CP Fax channels. Upon completion, the GDK System Service dependency list is updated to depend on the bus types detected in the system.

If the GDK service is not installed, **glHWDetect** will not return an error.

The glSystem data structure, unchanged from GDK 3.0, provides information about all working channels in the system. If this information is not desired, the first parameter may be NULL.

The glBoardSystem data structure, provides board-level resources and additional channel attributes. If this information is not desired, the second parameter may be NULL.

Example

The following are examples of valid calling conventions:

```
...
glSystem glSys;
glBoardSystem glBoard;
int retVal;

retVal = glHWDetect( &glSys, NULL, 0 );      // unchanged
retVal = glHWDetect( NULL, &glBoard, 0 );    // new usage
retVal = glHWDetect( &glSys, &glBoard, 0 );  // new usage
...
```

The glSystem and glBoardSystem data structures are defined in the include file *gfdboard.h*. More information about these structures follow:

glSystem

This structure contains identification information about the CP Fax Isa and Pci fax channels installed in the system.

```
struct _channel  {
       union {
              int    PortAddress;          //ISA only
              int    BoardNumber;          //PCI only
       }parm1;

       union {
              int    PhysicalChannel;      //ISA only
              int    ChannelOffset;        //PCI only
       }parm2;

       enum _bustype  BusType;

       int    BoardType;                    //type of fax channel
};
typedef struct _channel CHAN;

struct system  {
       CHAN Channel_ID[MAX_CHANNELS];       //length is numchan +
numBRI
       int numchan;
       int numBRI;
};
typedef struct system glSystem;
```

The glSystem structure has three fields; numchan, numBRI, and Channel_ID. The total of numchan and numBRI describe the number of valid elements in the Channel_ID array. The Channel_ID array elements contains channel specific information for each fax channel detected in the system.

Channel_ID elements [0 to numchan-1] describe fax channels; and elements [numchan to (numchan + numBRI) - 1] describe BRI controller cells.

NOTE: If numBRI is greater than zero, there will *always* be a CPi/200 BRI fax channel in the Channel_ID structure array.

Each Channel_ID element, CHAN, contains two union fields: parm1 and parm2. The valid fields within parm1 and parm2 are determined by the value of the BusType field as shown in the following chart.

BusType Value	Valid Parm Fields
gl_ISA	parm1.PortAddress
	parm2.PhysicalChannel
gl_PCI	parm1.BoardNumber
	parm2.ChannelOffset

The CHAN.BoardType member describes the model of the fax channel. This value is usually the same as the board model.

glBoardSystem

This structure contains detailed identification information about the CP Fax ISA and PCI fax boards and channels installed in the system.

```
typedef struct {
       unsigned char BoardType      : 4;
       unsigned char                : 1;
       unsigned char BoardRevision  : 2;
       unsigned char HasPollingBit  : 1;
} BoardInfo_t;

typedef struct {
       unsigned short CountryCode   : 10;
       unsigned short               : 6;
} BoardModel_t;

typedef struct {
       unsigned char ModemType      : 4;
       unsigned char NetworkIf      : 4;
} ModemInfo_t;

typedef struct {
       unsigned short Minor         : 8;
```

```
        unsigned short Major         : 8;
} RomVersion_t;

typedef struct {
        BoardInfo_t    BoardInfo;
        ModemInfo_t    ModemInfo;
        unsigned short OemStamp;
        unsigned char  CpuSpeed;
        unsigned char  Spare;
        BoardModel_t   BoardModel;
        RomVersion_t   RomVersion;
        char           RomVersionString[21];
        unsigned char  RomPersonality;
} RomId_t;

typedef enum _channelstate { gl_NotPresent, gl_Present };
typedef struct _channelex {
        int    glSystemChanIndex;

        union {
                struct {
                        int    PortAddress;          // ISA only
                        int    PhysicalChannel;      // ISA only
                } isa;

                struct {
                        int    BoardNumber;          // PCI only
                        int    ChannelOffset;        // PCI only
                } pci;
        } bus;

        int            BoardType;
        RomId_t        RomId;
        enum _channelstate State;
} Channel_t;

typedef struct _glResource {
        union {
                struct  {
                        int SHPortAddress;
                        int BRIPortAddress;
                        int BRIPhysicalChannel;
                } isa;
        } bus;
} GLResource_t;
```

```
typedef struct _board {
       int BoardModel;
       int CountryModel;
       Channel_t ChanList[MAX_CHANNELS_PER_BOARD];
       int NumChanCount;
       int BadChanCount;
       GLResource_t Resource;
       enum _bustype  BusType;
} Board_t;

typedef struct _boardsystem {
       int           numBoard;
       Board_t BoardID[MAX_BOARDS];
} glBoardSystem;
```

The glBoardSystem structure contains two data members, numBoard and BoardID. The value of numBoard describes the number of valid elements in the BoardID array. Each CP Fax physical hardware entity is represented by a BoardID element. For example, the CP4/LSI, a 4-channel analog board, is represented by one BoardID element.

The fields in BoardID represent board level information.

Board_t Fields	**Description**
BoardModel	Descriptive name of board. This value is usually the same as the value for Channel_t.BoardType, with the exception of CPD/220.
CountryModel	Country code retrieved from ROM that the board was certified for. If ROM does not contain this information, this value is zero.
ChanList	List of fax channels associated with physical board.
NumChanCount	The number of valid ChanList elements. It represents the total number of fax channels expected for BoardModel, i.e. a CP4/LSI is always exected to have four fax channels.

Board_t Fields	Description
BadChanCount	The number of undetectable fax channels for BoardModel. The range of this value is [0..NumChanCount].
Resource	Shared resource used by fax channels.
BusType	PC Bus interface type of BoardModel. Supported values are gl_ISA or gl_PCI.

The fields in BoardID.Resource represent board level resources, shared by the associated fax channels specified in the ChanList array.

Resource_t Fields	Description
bus.isa.SHPortAddress	I/O address of SC2000 resource, -1 if the resource does not exist.
bus.isa.BRIPortAddress	I/O address of BRI controller resource, -1 if the resource does not exist.
bus.isa.BRIPhysicalChannel	Physical channel of BRI controller resource, -1 if the resource does not exist.

The fields in BoardID.ChanList represent a fax channel.

Channel_t fields	Description
glSystemChanIndex	Zero-based index to glSystem structure. Value is -1 if the channel is not detected, i.e. State is gl_NotPresent.
bus.isa.PortAddress	For BoardID.BusType equal to gl_ISA, I/O address of ISA fax channel.
bus.isa.PhysicalChannel	For BoardID.BusType equal to gl_ISA, physical channel of ISA fax channel.
bus.pci.BoardNumber	For BoardID.BusType equal to gl_PCI, board id of PCI fax channel.
bus.pci.ChannelOffset	For BoardID.BusType equal to gl_PCI, PCI channel associated with board id.

Channel_t fields	Description
BoardType	Name of fax channel; it may be different from BoardID.BoardModel (i.e. CPD/220).
RomId	Raw identification information retrieved from fax channel's EPROM.
State	Indicates the fax channel's physical status. If the channel was detectable, the State is gl_Present, if not, the State is gl_NotPresent.

The fields in BoardID.ChanList.RomId represent the electronic signature of a fax channel.

RomId_t fields	Description
BoardInfo	Name of channel's board family and revision level.
ModemInfo	Identifies modem speed (i.e. 9600, 14400, etc.) and network interface type (i.e. digital or analog).
OemStamp	Value is always zero.
CpuSpeed	Speed of on-board processor.
Spare	Reserved.
BoardModel	Country code that board was certified for.
RomVersion	Version of Rom.
RomVersionString	Descriptive string containing version of Rom.
RomPersonality	Designates programmability of channel (i.e. toolkit vs. standard).

Example

```
#include <gfdboard.h>
void main( void )
{
   glSystem glSys = { 0 };
   glBoardSystem glBdSys = { 0 };
```

```
   int board = 0, chan = 0;
   int maxchan = 0;
   int retVal = 0;

   retVal = glHWDetect( &glSys, &glBdSys, 0 );
   if ( retVal == GL_HWCONFIG_SUCCESS )
   {
       // GLSYSTEM
       printf( "From glSystem structure:\n" );
       printf( "Detected %d fax channels and %d BRI resources.\n",
              glSys.numchan, glSys.numBRI );

       maxchan = glSys.numchan + glSys.numBRI;

       for ( chan = 0; chan < maxchan; ++chan )
       {
          if ( gl_ISA == glSys.Channel_ID[chan].BusType )
              {
                 printf( "Found Isa channel at PortAddress 0x%03x
             (%d)\n", glSys.Channel_ID[chan].parm1.PortAddress,

glSys.Channel_ID[chan].parm2.PhysicalChannel );
              }
                 else if ( gl_PCI ==
glSys.Channel_ID[chan].BusType )
                 {
                      printf( "Found Pci channel at Board:%X
Channel:%d\n",

       glSys.Channel_ID[chan].parm1.BoardNumber,

       glSys.Channel_ID[chan].parm2.ChannelOffset );
                 }
                 else
                      printf( "Unknown BusType %d\n",
glSys.Channel_ID[chan].BusType );
              }// end glSys channel loop

              // GLBOARDSYSTEM
              printf( "\nFrom glBoardSystem structure:\n" );
              printf( "Detected %d boards.\n", glBdSys.numBoard );

              // loop for board information
              for ( board = 0; board < glBdSys.numBoard; ++board )
              {
                 printf( "\nBoard %d - CountryModel is %d\n",
                         board + 1,
glBdSys.BoardID[board].CountryModel );
```

```
                // loop for channel information
                for ( chan = 0; chan <
glBdSys.BoardID[board].NumChanCount; ++chan )
                {
                    if ( gl_ISA == glBdSys.BoardID[board].BusType
)
                    {
                       printf( "Channel at PortAddress 0x%03x
(%d) is %s.\n",
glBdSys.BoardID[board].ChanList[chan].bus.isa.PortAddress,
glBdSys.BoardID[board].ChanList[chan].bus.isa.PhysicalChannel,
( gl_Present == glBdSys.BoardID[board].ChanList[chan].State
  ? "Present" : "Not Present" ) );
                    }
                    else if ( gl_PCI ==
glBdSys.BoardID[board].BusType )
                    {
                       printf( "Channel at Board:%X Channel:%d is
%s.\n",
       glBdSys.BoardID[board].ChanList[chan].bus.pci.BoardNumber,
       glBdSys.BoardID[board].ChanList[chan].bus.pci.ChannelOffset,
          ( gl_Present ==
glBdSys.BoardID[board].ChanList[chan].State
                ? "Present" : "Not Present" ) );
                    }
                    else
                    printf( "Unknown BusType %d\n",
                  glBdSys.BoardID[board].BusType );
                }// end glBdSys channel loop
            }// end glBdSys board loop
       }// end if glHWDetect successful
       else   {
           printf( "glHWDetect failed, error %d\n", retVal );
       }
}
```

Appendix A

Obtaining Additional Product Information

To obtain additional product information, visit the Dialogic web site at the following address:

www.dialogic.com

Technical Support

To contact Technical Support, visit this url:

http://support.dialogic.com

Appendix B
GFSH Commands

GFSH Utility

The primary function of the GFSH utility is to initialize the SC2000 chip, which is the CP Fax hardware interface to the data bus, in PEB or SCbus mode. All of the commands listed in Table 43 are used to configure GFSH.

The information in Table 43 includes the command names, parameters, and command descriptions.

Table 43. GFSH Commands

Command	Parameters	Description
setloc	<sh count><baseaddr>...<baseaddr9>	Initialize software with number and location of SC2000 chips
init	<index>	Initialize an SC2000 chip
updparm	<parm ID><value>	Modify certain SC2000 parameters
readparm	<parm ID><value>	Read current state of certain SC2000 parameters

Command	Parameters	Description
setbuscfg	<index><bus mode><bus speed> <clock speed> where <bus mode> is one of the following: “sc” or “0” “network” or “1” “resource” or “2” where <bus speed> is one of the following: “1.544” or “0” “2.048” or “1” “4.096” or “2” “8.192” or “3” (currently invalid) where <clock speed> is one of the following: “1.544” or “0” “2.048” or “1” “4.096” or “2” “8.192” or “3” (currently invalid) “16.384” or “4” (currently invalid) “32.768” or “5” (currently invalid)	Set SC2000 bus configuration parameters
connect	<index><local bus slot><type> <ext bus slot> where <type> is one of the following: “tx” or “0” “rx” or “1” “both” or “2”	Establish bus timeslot assignments

Command	**Parameters**	**Description**
disconnect	<index><local bus timeslot><type> where <type> is one of the following: “tx” or “0” “rx” or “1” “both” or “2”	Delete bus timeslot assignments
pebassert	<index><local bus timeslot><type>	Assert TSX (required after connect)
writefpga	<index><byte value>	Write FPGA configuration register bits
exit, quit, q, x		Exit this program
help, ?		Show usage
debug	0,1	Set debug level
echo	“on” | “off”	Enable/disable echo mode
verbose	“on” | “off”	Enable/disable verbose mode
save	GFAX	Saves the configuration file to the %GFAX% path

Appendix C
TIFF File Format

GDK maintains facsimile data in a TIFF (Tagged Image File Format) compression format. These files have a TIFF header of no fixed size that describes the data in the file, and helps to distinguish facsimile image files from other types of image files.

GDK uses a subset of the available set of image-description tags recommended in the Microsoft Version 6.0 TIFF specifications.

NOTE: GDK supports send-only faxing of multi-page TIFF files.

A TIFF file format offers these advantages:

- Facsimile data is stored in a compressed format.
- Graphics data from other sources can be used, after conversion.
- Files are compatible between GDK and printers, scanners, and other programs supporting TIFF.
- TIFF is an industry standard with acceptance from product developers.

GDK supports the following compression formats:

- TIFF Type 3
- TIFF Type 4

TIFF Type 3 is compatible with ITU-T Group 3 T.4 recommendation. Either 1-D or 2-D (two-dimensional) modified Huffman encoding compression can be used.

Each image line of a 1-D file is encoded as a TIFF Type 3 1-D line with EOL codes after each scan line, and the TIFF file is terminated with six EOLs to indicate the end of the page. The fill order can be MSBF (most significant bit first) or LSBF (least significant bit first).

The 2-D files also are called “Modified READ” (MR) files. The first image line of these files is encoded as a TIFF 3 1-D line. A certain number of lines, which is usually one to three, follow the 1-D line and encode only the differences between the current line and the previous one. Following the 2-D lines is another 1-D line. Then, there are more 2-D lines based on this new reference line. A flag that indicates whether a line is complete in itself (a 1-D encoded line) or is based on the preceding line (a 2-D encoded line) is embedded in the end of line code.

This encoding scheme results in approximately a 15 to 20 percent reduction in file size over 1-D encoding in most cases. This can vary greatly, depending on the type of image. A failure in one line affects only a small portion of the document.

TIFF Type 4 files also are called “Modified Modified READ” (MMR) files and require Error Correction Mode (ECM). This type is compatible with ITU Group 4 T.6 recommendation. An imaginary white line precedes the first line. Every line in the file is based on the differences between the current line and the line that preceded it; the first line presumes a blank line preceded it. This byte-oriented compression scheme results in approximately a 20 to 40 percent reduction in file size over one-dimensional encoding in most cases. A scan line error can corrupt an entire image from the point of the error forward.

Appendix D

Information for International Users

This appendix includes special information for Dialogic CP Fax customers calling from outside the United States or located outside North America.

> **CAUTION**
>
> In countries that do not have approval for the latest software, all features will not be available.

Full ASCII Character Set

To use the full character set during file conversion on the boards, add the following command to each channel in the GDK configuration:

GFXEXTEND 2

Country Codes

The international access codes for a number of countries are in the following table. These codes can also serve as values for the COUNTRY parameter in the registry.

Country	Code	Country	Code	Country	Code
Australia	61	Iceland	354	Norway	47
Austria	43	Indonesia	62	Poland	48
Bahrain	973	Israel	972	Portugal	351
Belgium	32	Italy	39	Singapore	65

Country	Code	Country	Code	Country	Code
Canada	1	Japan	81	South Africa	27
Chile	56	Jordan	962	Spain	34
Czech Republic	42	Korea	82	Sweden	46
Denmark	45	Luxembourg	352	Switzerland	41
Finland	358	Malaysia	60	Thailand	66
France	33	Mexico	52	Turkey	90
Germany	49	Netherlands	31	United Kingdom	44
Hong Kong	852	New Zealand	64	U.S.A.	1
Hungary	36				

Index

D

E

F

G

H

I

L

M

N

O

P

Q

R

S

T

U

W