HOST LINKS

GLAPI™

SDK

Application Programming Interfaces

GALLAGHER ROBERTSON

Version 6.4

GALLAGHER & ROBERTSON AS, Kongens gate 23, N- 0153 Oslo, Norway
Tel: +47 23357800 • Fax: +47 23357801
www: `http://www.gar.no/`
e-mail: support@gar.no

Contents

Contents

Host Links GlAPI

Installation

The G&R emulations and gateways are independent programs, but part of the `G&R Host Links` product set available on all major UNIX/Linux platforms. Many of the products are also available for Windows servers. For details on platforms supported, software delivery and installation refer to the *Host Links Installation and Configuration* manual.

Host Links product overview

Terminal environment

Host links products that run on UNIX or Linux servers with a terminal driven user interface include emulators and concentrators, as well as various utilities.

- G3270 provides synchronous IBM3270 functionality. G3270 emulates IBM LU type 2, including base and extended colour together with extended highlighting.

- Qsim provides synchronous Questar terminal functionality. Qsim simulates all Questar models, including the DKU7007, DKU7107, DKU7105 and DKU7211 (Mono, four colour A/B and seven colour modes are supported). It also simulates the VIP7760 and the VIP7700.

- V78sim provides Bull VIP78xx (BDS) functionality. V78sim emulates all models of the VIP7800 family; the actual reference is the BDS7. All visual attributes including colour are supported.

- Pthru provides transparent VIP7800 visibility to Bull mainframes for users with asynchronous VIP7800 terminals or emulators. The terminals are used in text or forms mode.

Server environment

Host Links products that run on UNIX, Linux or Windows servers.

- Ggate is a transparent gateway to the Bull native network. It is uniquely scalable in that it runs on UNIX/Linux and Windows servers. It avoids all need for Front-ends (MainWay/Datanet) or other gateways. It can be used to connect clients running G&R/Glink (for Windows or Java) emulators or any of the terminal emulators, terminal concentrators, network printer emulators and file transfer clients/servers in the Host Links product set. It also supports connections from third party clients using the Bull TNVIP and standard asynchronous Telnet protocols.

- Gweb provides a web browser interface to any host application that is otherwise accessible using the *Host Links Qsim*, *V78sim*, or *G3270* emulations.

- Gspool is designed to run as an unattended process and accept transparent print output from any type of host application (GCOS8, GCOS7, GCOS6, IBM) that normally sends print data to network printers (ROPs), or to a remote spooling system (DPF8-DS). On the Gspool system the print may be directed to a physical printer or to the local spooling system. Gspool operates in different modes, Connect mode, Terminal Writer mode, DPF8 mode, SNM mode, IBM mode, TN3270 mode and TN3270E mode.

- GUFT is an implementation of the Bull UFT file transfer protocols. These protocols enable transfer of data files between heterogeneous systems. The systems must be interconnected in a DSA/OSI network running over a private or public X25 network or over a local area network (LAN).

- Gproxy is a network management program used for supervision, management, load balancing and license sharing of G&R *Host Links* applications. Gproxy can be set up as a freestanding monitor program and/or report generator in a small network, or play a bigger role in a larger network.

Scope of the product

G&R/GlAPI is a set of Application Programmatic Interfaces to the *G&R/Gline* set of data communications line handlers. The API provides standard communications interfaces to data communication applications that is independent of the characteristics of the underlying network. The *Gline* line handlers support the following types of communications networks:

Asynchronous	Direct or modem connected
X.25	Native or PAD
TCP/IP	Dial-up, ISDN, leased line, Telnet, TN3270, TN5250, TNVIP
DSA/OSI	DSA and DSA/ISO Work Station (DIWS) over OSI networks
DSA/RFC1006	DSA and DSA/ISO Work Station (DIWS) over TCP/IP networks

GlAPI is used internally by all the communication application in the *G&R Host Links* product set. Several software development houses as well as individual data processing departments also use GlAPI when developing communications applications.

Run-time prerequisites

Any applications utilizing the G&R/GlAPI subroutine libraries requires the following G&R run-time packages to be installed in order to be able to execute.

All API's	*Gline run-time*.

The *Gline run-time* package is not required if your API programs are to connect via a *G&R/Ggate* transparent gateway which then takes care of the communication to the host system. Please note that if you already have other G&R communications or emulation packages installed such as *Ggate*, *Guft*, *Gweb*, *Qsim*, *G3270* etc., then the *Gline run-time* may already be installed.

The run-time pre-requisites for the various APIs are as follows:

Gline API	*Gline run-time only*
CPI-C API	*CPI-C run-time*
CPI-C 3270 API	*CPI-C 3270 run-time*
CPI-C 5250 API	*CPI-C 5250 run-time*
CPI-C 7800 API	*CPI-C 7800 run-time*
CPI-C DKU API	*CPI-C DKU run-time.*

Run-time licenses

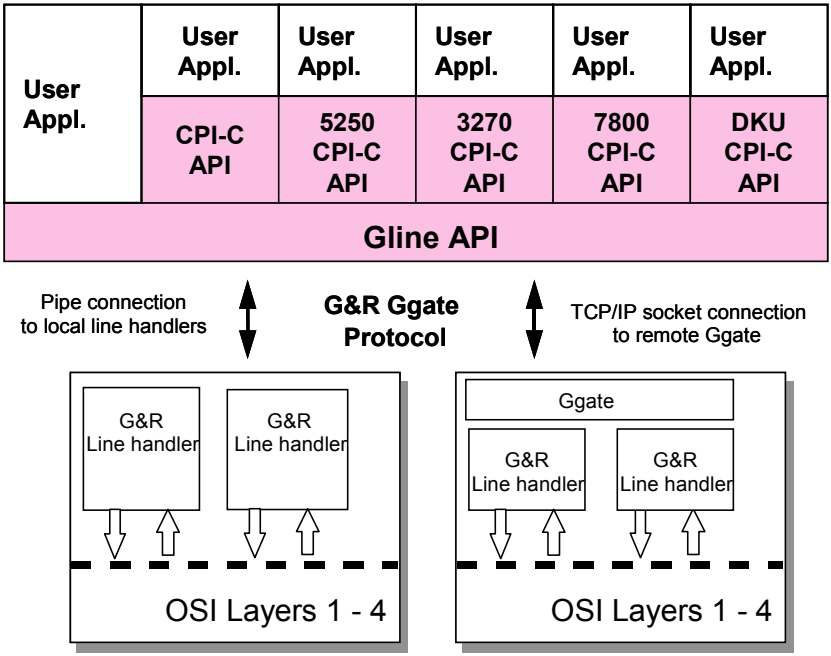
In order to run customer applications utilizing the G&R/GlAPI subroutine libraries, one or more of the following license keys must be present in your `/usr/gar/config/licenses` file:

License key	Description
`basic`	For the base G&R run-time system
`sdkglapi`	For GlAPI SDK
`glapi`	For GlAPI run-time

The licenses file identifies the G&R distributor, the owner of the license and the licensed products. The license key for a product will normally state how many users or simultaneous sessions the product is licensed for. If a limitation is specified in the license, only the licensed number of users or sessions can be active at any time.

GlAPI architecture

GlAPI provides six different programmatic interfaces: Gline API, CPI-C API, CPI-C 3270 API, CPI-C 5250 API, CPI-C 7800 API and CPI-C DKU API.



If the line handler runs on the same platform as the application using the Gline API, then the inter-process communication between the API and the line handler will be based on pipes. When connecting to Bull GCOS systems, the old front-ends require that the connection be made using DSA session over OSI-transport. If the network is router-based TCP/IP, you can place a `G&R/Ggate` gateway equipped with an OSI-stack at the central site, and connect via `Ggate` using the `G&R/Ggate` protocol over the TCP/IP network.

The GlAPI subroutine libraries

The UNIX *G&R Host Links GlAPI* subroutine libraries contain all the necessary interface routines that your application needs in order to be able to execute with their corresponding *GlAPI* run-time modules. The Windows version of the *GlAPI* subroutine libraries dynamically links to the run-time DLL's that contain the actual API subroutines.

The Gline API

This interface is used by all the applications in the *G&R Host Links* product set. It is a flexible and powerful interface consisting of a set of 'line services' that makes the characteristics of the underlying communication routines on the *Host Links* platforms transparent to the calling application. Line services are requested simply by calling them from the application program.

The CPI-C API

This interface operates on a higher level than the Gline APIs and is aimed at simple application-to-application transactional communication. It is a subset of the X/Open CPI-C/OSI primitives set and provides the necessary functions in order to connect, disconnect, send and receive data to/from a host application.

The CPI-C 3270 API

This interface is a programmatic interface to the G&R IBM3270 emulation called `G3270`. It provides a subset of X/Open CPI-C primitives just as the CPI-C API does but in this case using IBM3270 presentation. Data from the host system is processed by the IBM3270 emulation routines and delivered by the API to the application in a virtual screen format. The application passes data to the API in a keyboard buffer format. The emulation is handled completely by the API and no control sequences are delivered to the application. All function keys, including send- and edit-keys, are supported.

The CPI-C 5250 API

This interface is a programmatic interface to the G&R IBM5250 emulation called `G5250`. It provides a subset of X/Open CPI-C primitives just as the CPI-C API does but in this case using IBM5250 presentation. Data from the host system is processed by the IBM5250 emulation routines and delivered by the API to the application in a virtual screen format. The application passes data to the API in a keyboard buffer format. The emulation is handled completely by the API and no control sequences are delivered to the application. All function keys, including send- and edit-keys, are supported.

The CPI-C 7800 API

This interface is a programmatic interface to the G&R VIP7800 emulation called `V78sim`. It provides a subset of X/Open CPI-C primitives just as the CPI-C API does but in this case using VIP7800 presentation. Data from the host system is processed by the VIP7800 emulation routines and delivered by the API to the application in a virtual screen format. The application passes data to the API in a keyboard buffer format. The emulation is handled completely by the API and no control sequences are delivered to the application. All function keys, including send- and edit-keys, are supported.

The CPI-C DKU API

This interface is a programmatic interface to the G&R Questar, VIP7700 and VIP7760 emulation called *Qsim*. It provides a subset of X/Open CPI-C primitives just as the CPI-C API does but in this case using DKU, VIP7700 or VIP7760 presentation. Data from the host system is processed by the emulation routines and delivered by the API to the application in a virtual screen format. The application passes data to the API in a keyboard buffer format. The emulation is handled completely by the API and no control sequences are delivered to the application. All function keys, including send- and edit-keys, are supported.

NOTE
Any application using GlAPI can only be used in conjunction with their corresponding GlAPI or emulation CPI-C run-time module. These run-time modules must be installed when running the application. The *G&R/GlAPI Run-time* license covers the total number of concurrent sessions of all APIs in use.

Files delivered with GlAPI

`glapi.h`	Include file. To be included in all applications using the Gline API library.
`cpic.h`	Include file. To be included in all applications using any of the CPI-C API libraries.
`apitest.c`	Example Gline API program using one logical line.
`apitest2.c`	Example Gline API program using two logical lines.
`apiclnt.c`	Example Gline API program using one logical line.
`apiserv.c`	Example Gline API program using two logical lines.
`cpicline.c`	Example CPI-C API program connecting to TSS.
`cpictest.c`	Example CPI-C 3270 API program.
`cpictst.pl`	Example CPI-C API program in Perl
`cpicclnt.c`	Example CPI-C API client program connecting to `cpicserv.c`
`cpicserv.c`	Example CPI-C API server program accepting connection from `cpicclnt.c`
`dkuiof.c`	Example CPI-C DKU API program for GCOS7 access
`dkutss.c`	Example CPI-C DKU API program for GCOS8 access
`glapitst.pl`	Example Gline API program in Perl

UNIX only

`libglapi.a`	Object library of Gline line handler API functions.
`libglcpic.a`	Object library of CPI-C line handler API functions.
`libgl3270.a`	Object library of CPI-C 3270 API functions.
`libgl5250.a`	Object library of CPI-C 5250 API functions.
`libgl7800.a`	Object library of CPI-C 7800 API functions

`libglqsim.a`	Object library of CPI-C DKU API functions.
`Makefile`	Makefile for compilation of example programs.

Windows only

`glapi.hlp` `glapi.cnt`	GlAPI on-line help
`vc32.bat`	Batch file to compile and link the test programs with Microsoft C 32-bit compilers
`wincpic.h`	Include file. To be included in all applications using any of the Windows CPI-C API functions.
`glapim32.lib`	VC++ static link library to access `glapi32.dll`
`gcpicm32.lib`	VC++ static link library to access `wcpic32.dll`
`g3270m32.lib`	VC++ static link library to access `cpic3270.dll`
`g5250m32.lib`	VC++ static link library to access `cpic5250.dll`
`g7800m32.lib`	VC++ static link library to access `cpic7800.dll`
`gdkum32.lib`	VC++ static link library to access `cpicdku.dll`

Perl interface modules

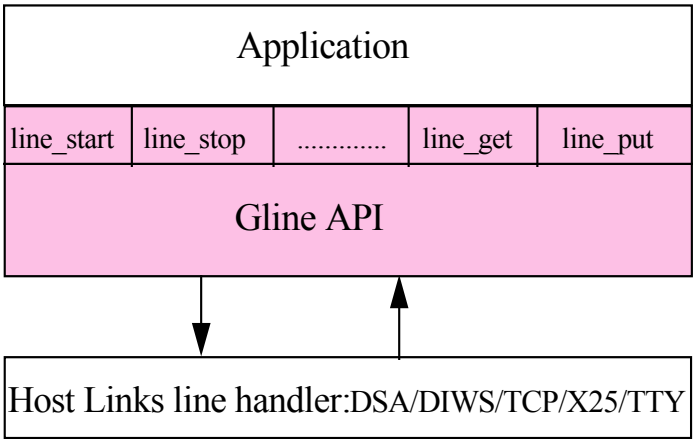
`Glapi.pm` `Glapi.so/.dll`	Gline line handler API functions Perl module and shared library
`Cpic.pm` `Cpic.so/.dll`	CPI-C line handler API functions Perl module module and shared library
`Cpic3270.pm` `Cpic3270.so/.dll`	CPI-C 3270 API functions Perl module module and shared library
`Cpic5250.pm` `Cpic5250.so/.dll`	CPI-C 5250 API functions Perl module module and shared library
`Cpic7800.pm` `Cpic7800.so/.dll`	CPI-C 7800 API functions Perl module module and shared library

`CpicDku.pm`	CPI-C DKU API functions Perl module module and shared library
`CpicDku.so/.dll`	

NOTE
The `cpic.cfg` configuration example and the below DLL's are delivered with the *`GlAPI Run-time`* package and installed in the `c:\gar\bin32` directory.

`glapi32.dll`	32-bit Gline API DLL.
`wcpic32.dll`	32-bit CPI-C DLL.
`cpic3270.dll`	32-bit CPI-C 3270 DLL
`cpic5250.dll`	32-bit CPI-C 5250 DLL
`cpic7800.dll`	32-bit CPI-C 7800 DLL
`cpicdku.dll`	32-bit CPI-C DKU DLL
`gapidb32.dll`	32-bit Gline API DLL for debugging

Gline API



Gline API Programming Issues

Architecture

An application using the Gline API will at run-time connect to a Gline line handler module. The application and the line handler are separate executables. The first call an application should use is `line_initialize()`, and the last, `line_release()`.

The application starts the line handler with a function call, `line_start()`

`line_start():`

- Starts a line handler based on the parameters previously delivered to the API by calls to `line_init_parameter()`.
- The line handler runs as a separate process and may send and receive data to and from the host while the application carries out other tasks.
- If the line handler is a local one, i.e. runs on the same system as the application, then two pipes will be created for communication between the application and the line handler.
- If the line handler is a remote one, i.e. communicating through a *Ggate* gateway running on another system, then a TCP/IP socket is created and connection to *Ggate* is established. *Ggate* then starts the line handler. Communication between the application and the line handler is then carried out over the TCP/IP socket.
- The line handler can be one of the following:

Line handler	Communication type	Comms. API
`gl_dsa`	DSA or DSA/ISO WorkStation (DIWS) session over OSI transport	XTI, TLI
	DSA or DSA/ISO WorkStation (DIWS) session over RFC1006 (G&R OSI TP0 over TCP/IP)	Sockets
`gl_tcp`	'Raw' TCP/IP	Sockets
	Telnet	Sockets
	TNVIP client	Sockets
	TN3270 client	Sockets
	TN5250 client	Sockets
`gl_x25`	PAD	Provider specific
	'Raw' X25	Provider specific
`gl_tty`	Asynchronous line	Basic OS

The line handler is chosen by giving one of the following parameters to Gline: `-LI DSA`, `-LI DIWS`, `-LI TCP`, `-LI X25` or `-LI TTY`.

A remote line handler is chosen by giving a parameter in the form `-LI DSA:gateway` where 'gateway' is the symbolic name or IP address of the *Ggate* system. Other than the form of the parameter used to start the remote line handler there is absolutely no difference between using a local or a remote line handler.

Several line handlers may be started by the application. A unique logical line identifier (LID) is associated with each line to distinguish one line from another.

Exporting and importing sessions between applications

The Gline API allows you to pass sessions between applications. This is done via the `line_stopexport()` function which returns a session identifier which may then be used in the `line_startimport()` function in another program.

Programming in a Windows multi-threaded environment

`line_initialize()/line_release()` only needs to be called once per application and must be the first and last functions called.

The DLL is not completely thread-safe, however with a few precautions you can happily program multiple threads accessing multiple GlAPI session line ids (lid). You should never have multiple threads accessing the same lid simultaneously.

The lids are indexes to a table and there is a 'static' internal variable that contains the value of the current lid. The `line_switch()` function can be used to select any specific lid.

This current lid value obviously gets changed when the `line_switch()` function is called, so there is conflict if a second thread calls this function before the first thread has called the GlAPI function it required.

Each lid has all it's necessary internal variables in an allocated structure amongst which are the buffers returned by the `line_get()` function, so no buffer copying is required.

The line functions are not very CPU consuming, and apart from startup `line_start()` and receive `line_get()` they are not blocking.

The receive functions are only blocking if there is no data, so only calling them if data is available solves that function. `line_input_available()`, `line_wait_lid()` or `line_wait()` can be used to check for incoming data. As `line_wait_lid()` gets supplied the lid value, this is thread safe and therefore no semaphore will be needed.

So the above means that you can write multi-thread programs with the small restriction that GlAPI functions should not be called simultaneously. As they are not time-consuming, then this should not hinder execution speed nor cause a bottle neck.

The following code is none time consuming and safe:

```
void a_thread_function(int lid)
{
    /* not need for thread semaphor here */
    if (line_wait_lid(lid, 200) == lid) {
        /* now we need a semaphore */
        reserve_glapi_mutex()
        line_switch(lid)
        line_get(...)
        line_put(...)
        release_glapi_mutex()
    }
}
```

You will need to use the `line_switch()` before every function which does not take lid as a parameter.

If you're program is event driven, either by message or timer, then it could also do something like this:

```
void a_thread_event(int lid)
{
    /* we need a semaphore */
    reserve_glapi_mutex()
    line_switch(lid)
    if (line_input_available()) {
        line_get(...)
        line_put(...)
    }
    release_glapi_mutex()
}
```

The remaining problem is the `line_start()` function. This function may be slightly time consuming. It will be more noticeable for Ggate connections over the LAN than local connections to the DSA/DIWS G&R Listener (nl_dsa) line handlers.

The only 100% safe solution for `line_start()` is to block the GlAPI during the whole `line_start()` function. However, the first thing this function does is to pick up the lid pointer from the current lid value, once it has this pointer, the rest of the function including the connection is thread safe. This means that in most situations it would reasonably safe to have a separate mutex for `line_start()` and only wait for the mutex for a short time. This obviously isn't 100% safe.

Another few points of interest are that the line variables:

```
extern char line_error[LINEERR_SIZE + 1];
extern char line_function[2];
extern int  line_device_status;
extern int  line_device_address;
```

These are not thread safe, so you would need to copy them if needed before you release the your GlAPI mutex. You only need to copy the line_error[] variable if a line function reports an actual error.

Gline API variables and definitions

external variables

The Gline API supplies several variables for general information.

SYNOPSIS (C)

```
#define LINEERR_SIZE     200
extern int          line_device_status;
extern int          line_device_address;
extern char         line_function[2];
extern char         line_error[LINEERR_SIZE + 1];
```

SYNOPSIS (PERL)

```
($status, $address, $func1, $func2) = Glapi::VIPHeader();
$error_text_string = Glapi::Errmsg();
```

definitions

SYNOPSIS (C)

```
#define true  1
#define TRUE  1
#define false 0
#define FALSE 0
typedef char boolean;
```

enclosure_t type

SYNOPSIS (C)

```
typedef enum { e_none, e_segment, e_message,
e_group, e_turn, e_unused, e_attmsg } enclosure_t;
```

SYNOPSIS (PERL)

```
$Glapi::e_none, $Glapi::e_segment, $Glapi::e_message,
$Glapi::e_group, $Glapi::e_turn, $Glapi::e_attmsg
```

Enclosure	Description
`e_none`	No enclosure. Internal transfer between line module and application.
`e_segment`	End of segment. Returned when the host has split the record into several segments and this is not the last one. The host keeps the turn.
`e_message`	End of message. Returned when the host is sending several records (messages) and this is not the last one. The host keeps the turn.
`e_group`	End of message group. Returned when the host sends the final record (message) and gives the application the turn. Hosts using DSA session may send one or several records (messages, each split into several segments) before giving the turn to the application.
`e_attmsg`	Attention message from line module. Returned when the line module reports different kinds of events. If `line_function[1] == 0`, it is an informational message, if `!= 0`, it is an error message. Note that an error on output (`line_put()`) will be reported on the next call to `line_get()`. After an attention message has been received, you can use the function `line_connected()` to check if the connection was taken up or down.

Gline API functions list

Function	Prototype
initializes the line handler API	`boolean line_initialize();`
terminates the line handler API	`void line_release();`
get logical line id (LID)	`boolean line_get_lid();`
change logical line (LID)	`void line_switch();`
Obsolete function to provide line handler startup parameters	`int line_init_params();`
Give line handler parameters	`void line_parameter();`
Start line handler	`int line_start();`
Imports a session from a line module	`int line_startimport();`
stop line handler	`void line_stop();`
exports the line module session	`void line_stopexport();`
stop line handler, but keep all line parameters given for this LID	`void line_stopkeep();`
collect input from line	`int line_get();`
insert data at the beginning of the line buffer	`boolean line_unget();`
write data to line	`void line_put();`
write character to line	`void line_putc();`
write string to line	`void line_puts();`
write buffer to line	`void line_write();`
wait for line input	`int line_wait();`
wait for lid line input	`int line_wait_lid();`
check if input on line	`boolean line_input_available();`
check whether line is up or not	`boolean line_connected();`

Function	Prototype
check if it is our turn	`boolean line_our_turn();`
check for two-way simultaneous session	`boolean line_simultaneous();`
demand turn from line	`void line_demand_turn();`
check status of logical line id (LID).	`void line_select();`
sets line_wait() callback function.	`void line_waitcallback();`
Initiates keep alive logic	`void line_init_keepalive();`
Sends keep alive packet	`void line_send_keepalive();`

Gline API functions

line_initialize

NAME

`line_initialize` - initializes the line handler API

SYNOPSIS (C)

```
boolean GLAPI line_initialize (
        const char * prog,
        int          id,
        const char * ver);
```

SYNOPSIS (PERL)

```
$rc = Glapi::Initialize($prog, $id, $ver);
```

DESCRIPTION

This function must be the first function called before using any other of the line handler API calls. This function allocates internal memory necessary for the GlAPI. The `prog` and `ver` parameters must be NULL, and the `ver` must be ZERO.

If the API is being called by a DLL, then the DLL must call this function once for each application. If the API is being called from a DDL that needs to share sessions between applications, it must link the DLL to the `glapd.lib` library that in turn loads the `glapd.dll`. Session sharing is restricted to Windows 16-bit applications only.

RETURN VALUE

Returns false on failure, otherwise true is returned on success.

SEE ALSO

`line_release()`

line_release

NAME

`line_release` - terminates the line handler API

SYNOPSIS (C)

```
void GLAPI line_release (void);
```

SYNOPSIS (PERL)

```
Glapi::Release();
```

DESCRIPTION

This function must be the last function called to the line handler API. This function liberates any internal memory.

If the API is being called by a DLL, then the DLL must call this function once for each application.

RETURN VALUE

No return value.

SEE ALSO

`line_initialize()`

line_get_lid

NAME

`line_get_lid` - get an available line id

SYNOPSIS (C)

```
boolean GLAPI line_get_lid (
        int  * lid);
```

SYNOPSIS (PERL)

```
$lid = Glapi::Get_lid();
```

DESCRIPTION

When the application starts and uses several lines, this function can be used to ensure unique identification of the different lines. The application will propose a line id value in `*lid`, and `line_get_lid()` will check whether it is available or not. If it is, the same value is returned in `*lid`. Otherwise `line_get_lid()` will return another valid lid in `*lid`. The function will use the value in `*lid` as a starting point and search upwards for an available lid. The line id can take values from 0 to 63.

RETURN VALUE

A value of 1 (true) is returned when an available lid was found. This lid could be the value suggested or the next available. When no lid is available between the suggested value and 63, the function returns the value of 0 (false).

SEE ALSO

`line_switch()`

line_switch

NAME

`line_switch` - switch to another line

SYNOPSIS (C)

```
void GLAPI line_switch (
        int    lid);
```

SYNOPSIS (PERL)

```
Glapi::Switch($lid);
```

DESCRIPTION

The application may start and use several lines. To distinguish one line from another, a unique logical line identifier (lid) must be chosen for each line started. From the applications point of view, the only difference between these lines is the lid value. To address a specific line (lid), the application must first switch to that lid with the `line_switch(lid)` call. Subsequent calls to the line module will address that lid until a new `line_switch(lid)` call is issued.

For applications using one line only, the `line_switch(lid)` call may be omitted, and the default lid value of zero will then be used. If `line_get_lid()` has been called, to get a valid lid, the application should also call `line_switch()` with that lid.

RETURN VALUE

No return value.

SEE ALSO

`line_get_lid()`

line_init_params

NAME

`line_init_params` - set up line parameters (before starting it)

SYNOPSIS (C)

```
int GLAPI line_init_params (
        char * parameter);
```

SYNOPSIS (PERL)

```
$rc = Glapi::Init_params($parameter);
```

DESCRIPTION

Before issuing the `line_start()` call, the application must specify which line module to start. This can be done by the use of `line_init_params()`, but this function has now become obsolete since all line parameters can be delivered by the `line_parameter()` function. The supplied parameters will be saved and can be used for the next `line_start()` call for the same lid. The actual parameter values are kept when `line_stopkeep()` is used, and released when `line_stop()` is called.

Examples:

```
line_init_params("-LI");
line_init_params("DSA");
line_init_params("-MN");
line_init_params("TEST");
```

RETURN VALUE

Upon successful completion, a value of 0 is returned. Otherwise, a value of -1 is returned. The application should terminate if the value is not zero.

SEE ALSO

`line_parameter()`

line_parameter

NAME

`line_parameter` - send parameter to the line module (after startup)

SYNOPSIS (C)

```
void GLAPI line_parameter (
        char * parameter_type,
        char * parameter_value);
```

SYNOPSIS (PERL)

```
Glapi::Parameter($parameterType, $parameterValue);
```

DESCRIPTION

This function accepts any line control parameter and delivers it to the line module. This function can be used at any time, and is used to change a parameter in the line module. This function should be used in all new applications to set initial line handler parameters and to change its setting after startup.

The function reads the parameter type and the parameter value as to separate strings, i.e.

```
line_parameter("-HM", "DPS8");
```

RETURN VALUE

A boolean value of 1 (true) is returned if the function has been performed. A value of 0 (false) is returned if the function cannot be performed

SEE ALSO

`line_init_params()`

line_start

NAME

`line_start` - start the line module

SYNOPSIS (C)

```
int GLAPI line_start(void);
```

SYNOPSIS (PERL)

```
$rc = Glapi::Start();
```

DESCRIPTION

This call starts the specified line handler as a new process. This process will remain active until `line_stop()` or `line_stopkeep()` is called. In test situations this means that the process may live after the application program has aborted. The application should use `line_get_lid()` and/or `line_switch()` to set up the correct line number if more than one line module is started. The `line_parameter()` function must have been called at least once in order to identify the line handler to be started.

RETURN VALUE

Upon successful completion, a value of 0 is returned. Otherwise, a value different from zero is returned.

ERRORS

If `line_start()` fails to start the specified line handler, it returns a value different from 0 and `line_error` will contain a message describing the reason.

SEE ALSO

`line_startimport()`, `line_get_lid()`, `line_switch()`, `line_parameter()`, `line_stop()`, `line_stopexport()`, `line_stopkeep()`

line_startimport

NAME

`line_startimport` – imports a session from a line module

SYNOPSIS (C)

```
int GLAPI line_startimport(
        char * export_name);
```

SYNOPSIS (PERL)

```
$rc = Glapi::Startimport($exportName);
```

DESCRIPTION

This call imports a session from an existing line handler process. The `export_name` input parameter must contain a valid session identifier returned from the `line_stopexport()` function. On success, the imported session will be in the same state as when the previous process exported it.

RETURN VALUE

Upon successful completion, a value of 0 is returned. Otherwise, a value different from zero is returned.

ERRORS

If `line_startimport()` fails to connect to the specified line handler, it returns a value different from 0 and `line_error` will contain a message describing the reason. This function will fail if the line handler has terminated due to the timeout supplied in the `line_stopexport()` function.

SEE ALSO

`line_start()`, `line_stop()`, `line_stopexport()`, `line_stopkeep()`

line_stop

NAME

`line_stop` - stop line module

SYNOPSIS (C)

```
void GLAPI line_stop(void);
```

SYNOPSIS (PERL)

```
Glapi::Stop();
```

DESCRIPTION

This function should be called when the host session on a selected line has been closed and the application wishes to stop the line module. It must be called before terminating the application. It will free local memory used for startup parameters. The program should use the `line_switch()` function to set up the correct line number if more than one line module has been started.

RETURN VALUE

No return value.

SEE ALSO

`line_start()`, `line_startimport()`, `line_stopexport()`, `line_stopkeep()`

line_stopexport

NAME

`line_stopexport` – exports the line module session

SYNOPSIS (C)

```
int GLAPI line_stopexport (
        int    timeout,
        char * export_name,
        int    buflen);
```

SYNOPSIS (PERL)

```
($exportName, $rc) = Glapi::Stopexport($timeout);
```

DESCRIPTION

This function is similar to calling `line_stop()` function except that the line module is not terminated. A session identifier is returned in the `export_name` buffer. This identifier may then be used by another process to import the session using `line_startimport()`.The `timeout` parameter specifies the number of seconds the line module should wait for a process to import the session. If this timeout expires, the line module will automatically close the session and terminate. The `buflen` parameter must be initialized with the size of the `export_name` buffer length. The returned `export_name` is a normal null terminated string.

RETURN VALUE

The return value is the length of the string returned in the `export_name` variable. A return value of 0 indicates an error.

ERRORS

A return value of 0 indicates an error, and `line_error` will contain a message describing the reason.

SEE ALSO

```
line_start(), line_startimport(), line_stop(),
line_stopkeep()
```

line_stopkeep

NAME

`line_stopkeep` - stop line module

SYNOPSIS (C)

```
void GLAPI line_stopkeep(void);
```

SYNOPSIS (PERL)

```
Glapi::Stopkeep();
```

DESCRIPTION

This function should be called when the host session on a selected line has been closed and the application wishes to stop the line module. It must be called before terminating the application. Local memory used for startup parameters will be kept. This means that any parameters passed using `line_parameter()` and `line_init_params()` calls before `line_start()` will be available the next time the application issues a call to the `line_start()` function. The program should use the `line_switch()` function to set up the correct line number if more than one line module has been started.

RETURN VALUE

No return value.

SEE ALSO

`line_start()`, `line_startimport()`, `line_stop()`, `line_stopexport()`

line_get

NAME

`line_get` - collect input from line

SYNOPSIS (C)

```
int GLAPI line_get (
        char       ** buffer,
        enclosure_t * enclosure);
```

SYNOPSIS (PERL)

```
($buffer, $enclosure) = Glapi::Get();
```

DESCRIPTION

This call should be used when the application expects data from the host. If no input is available, the application will be suspended until it is. All error reporting from the line module is done through this call, including errors from previous send.

Note that the application may receive a remote connect in addition to remote data. A remote connect is presented to the application as an attention message with `line_function[1] == 0`. The application may use the `line_wait()` call to wait for a remote connect. If the application has started several line modules, the `line_wait()` may be useful as it reports back the lid for which input is available.

The number of characters available at the address given by `*buffer` is given as the return value of `line_get()`. The enclosure level is delivered in `*enclosure`, which can take the values defined in the table on page 20.

The external variables `line_device_address`, `line_function[]`, and `line_device_status` will contain the VIP Line Protocol Header if supported by the line protocol used. Otherwise these variables should be ignored.

It is important to note that the line module notifies the application only when delivering the first message block: after a call to `line_get()`, the application should always call `line_input_available()` to check if all available data has been read. If the application receives data on a line without reading it, a subsequent call to `line_wait()` may not "see" these data. `line_wait()` will work reliably only if the application always reads all the data from the line buffers. You should also note that when the application has made a call to `line_unget()` since the previous call to `line_get()` the next `line_get()` will return immediately without waiting for new data from the line handler. The line buffer will then contain the unget data followed by any new data available from the line handler.

RETURN VALUE

The number of characters available in the buffer is returned

SEE ALSO

`line_unget(), line_input_available()`

line_unget

NAME

`line_unget` - insert data at the beginning of the line buffer

SYNOPSIS (C)

```
boolean GLAPI line_unget (
        char * buffer,
        int    size);
```

SYNOPSIS (PERL)

```
$rc = Glapi::Unget($buffer,$size);
```

DESCRIPTION

This function allows the application to insert data in the line handler's input buffer. The inserted unget data will be delivered to the application as the first `size` bytes of data in the line buffer when it makes the next call to `line_get()`. Note that only one call to `line_unget()` is allowed before reading the data with `line_get()`. Calling this function with `size` 0, will clear the unget buffer. You should also note that after a call to `line_unget()`, `line_get()` will not block waiting for new data from the line handler, but return the unget data. `line_input_available()` or `line_wait()` should be used to wait for new data.

The `line_unget()` function is typically used when the buffer delivered by `line_get()` ends with an incomplete control sequence or command and you want to delay the interpretation of the rest of the buffer until the rest of the sequence or command is available.

RETURN VALUE

A value of 1 (true) is returned if the `line_unget()` function has been performed. A value of 0 (false) is returned if the `line_unget()` function cannot be performed for reasons like: a `line_unget()` operation has already been performed, the size is greater than the maximum unget size (80 bytes) or the size is less than 0.

SEE ALSO

`line_get(), line_input_available()`

line_put

NAME

`line_put` - write data to the line

SYNOPSIS (C)

```
void GLAPI line_put (
        char      * buffer,
        int         buffer_length,
        enclosure_t enclosure);
```

SYNOPSIS (PERL)

```
Glapi::Put($buffer, $length, $enclosure);
```

DESCRIPTION

This function is used for sending text to the host, and to issue commands to the line modules.

The line module interpret some text strings as special commands:

`$*$CN [parameters]`	Connect to a remote system.
`$*$DIS`	Disconnect an established connection.
`$*$BRK`	Send break/attention message to host.

Users familiar with other Gallagher & Robertson communication products will find that the same format is used in the end user applications. See the line handler sections for further details on this.

If the remote side is initiating the session, the addressed GLAPI application will be notified of the incoming session connect request by an attention message. For more information see the `line_get()` call on page 36.

The connect request may be refused for all of the normal reasons. The error will be reported through the `line_get()` call, because an error on send will normally not be detected until some time later.

This philosophy is used for all send errors, also those occurring when sending normal application text. The application program must be programmed accordingly. This means that for a normal two-way alternate session the application must issue a receive request after a send, and will receive either text or an error message from the send. If the program is intended to work asynchronously, i.e. carry out other tasks while waiting for a host reply, calls to `line_input_available()` can be used to test if input is available.

Two-way simultaneous (data collection) applications must be programmed accordingly. Since the host is not required to answer input, the program MUST NOT do a receive until input is available.

This function reads three parameters: `buffer`, `buffer_length` and `enclosure`. "`buffer`" holds the address of the first character to be sent, and "`buffer_length`" the number of characters to be sent. Refer to the `line_write()` documentation on page 44 for the meaning of the "`enclosure`" variable.

The external variables `line_function[]`, `line_device_address` and `line_device_status` will contain the VIP Line Protocol Header if supported by the line protocol used. Otherwise these variables should be ignored.

RETURN VALUE

The function returns an integer. If positive it is the number of characters written to the line. The function can return a negative integer indicating an error:

-1 - internal error (line map not found)
-2 - buffer too big (>64KB)
-3 - can't allocate buffer space (out of memory)

SEE ALSO

`line_putc(), line_puts(), line_write(), line_our_turn(), line_simultaneous()`

line_putc

NAME

`line_putc` - write a character to the line

SYNOPSIS (C)

```
void GLAPI line_putc (
        char    c);
```

SYNOPSIS (PERL)

```
Glapi::Putc($char);
```

DESCRIPTION

Move one byte to the line module buffer. The buffered data will not be sent to the host until a `line_write()` or `line_put()` that flushes the data is called.

In this way it is possible to build up the send buffer one character at a time before actually sending it to the host.

RETURN VALUE

The function returns an integer. If positive it is the number of characters written to the line. The function can return a negative integer indicating an error:

-1 - internal error (line map not found)
-2 - buffer too big (>64KB)
-3 - can't allocate buffer space (out of memory)

SEE ALSO

`line_put()`, `line_puts()`, `line_write()`, `line_our_turn()`, `line_simultaneous()`

line_puts

NAME

`line_puts` - write a string to the line

SYNOPSIS (C)

```
void GLAPI line_puts (
        char * buffer,
        int    bufferlength);
```

SYNOPSIS (PERL)

```
Glapi::Puts($buffer, $length);
```

DESCRIPTION

Move a string to the line module buffer. The buffered data will not be sent to the host until a `line_write()` or `line_put()` that flushes the data is called.

In this way it is possible to build up the send buffer one string at a time before actually sending it to the host.

RETURN VALUE

The function returns an integer. If positive it is the number of characters written to the line. The function can return a negative integer indicating an error:

-1 - internal error (line map not found)
-2 - buffer too big (>64KB)
-3 - can't allocate buffer space (out of memory)

SEE ALSO

`line_put()`, `line_putc()`, `line_write()`, `line_our_turn()`, `line_simultaneous()`

line_write

NAME

`line_write` - write data to line

SYNOPSIS (C)

```
void GLAPI line_write (
        enclosure_t enclosure);
```

SYNOPSIS (PERL)

```
Glapi::Write($enclosure);
```

DESCRIPTION

This function forces a write of the current buffer to the line. The parameter, `enclosure`, specifies the enclosure level, and can take the values defined in the table on page 20.

RETURN VALUE

No return value.

SEE ALSO

`line_put(), line_putc(), line_puts()`

line_wait

NAME

`line_wait` - wait for line input

SYNOPSIS (C)

```
int GLAPI line_wait (
        int    time-out);
```

SYNOPSIS (PERL)

```
$lid = Glapi::Wait($timeout);
```

DESCRIPTION

This call may be used by an application that wishes to wait for any line input within a specified time period. This call may be very useful if the application has started several line modules. Upon return from this call, the application will be notified which line, if any, has line input. The time-out interval is given in seconds.

It is important to note that this call will suspend the application in a passive wait.

In the UNIX version it can however be interrupted by signals and it will return 99 when a signal has interrupted its execution.

In the Windows version, the `line_waitcallback()` function can be used to set up a function that will be called to handle Windows messages received while in the `line_wait()` function. If the callback function returns FALSE after handling a Windows message, the `line_wait()` will exit with the return value of 99.

It is the applications responsibility to recognize whether `line_wait()` was interrupted by the above or not.

If there is data available when this function is called, it will return immediately.

RETURN VALUE

The function will return the logical identifier (lid) of the line module having input available. If time-out, -1 will be returned. Return value of 99 indicates that an error has occurred.

ERRORS

If 99 is returned if a fatal error occurred and the application should call `line_stop()` or `line_stopkeep()`, unless the function has been interrupted under your own control by a signal or callback.

SEE ALSO

`line_get(), line_input_available()`

line_wait_lid

NAME

`line_wait_lid` - wait for line input from one lid

SYNOPSIS (C)

```
int GLAPI line_wait_lid (
        int    lid,
        int    time-out);
```

SYNOPSIS (PERL)

```
$lid = Glapi::Wait_lid($lid, $timeout);
```

DESCRIPTION

This call may be used by an application that wishes to wait for line input on a specific lid within a specified time period. This call may be very useful if the application has started several line modules but is interested in input from a specific one of them. Upon return from this call, the application will be notified if the lid has line input. The time-out period is specified by a number of seconds.

It is important to note that this call will suspend the application in a passive wait.

In the UNIX version it can however be interrupted by signals and it will return 99 when a signal has interrupted its execution.

In the Windows version, the `line_waitcallback()` function can be used to set up a function that will be called to handle Windows messages received while in the `line_wait_lid()` function. If the callback function returns FALSE after handling a Windows message, the `line_wait_lid()` will exit with the return value of 99.

It is the applications responsibility to recognize whether `line_wait_lid()` was interrupted by the above or not.

If there is data available when this function is called, it will return immediately.

RETURN VALUE

The function will return the logical identifier (lid) of the line module having input available. If time-out, -1 will be returned. Return value of 99 indicates that an error has occurred.

ERRORS

If 99 is returned if a fatal error occurred and the application should call `line_stop()` or `line_stopkeep()`, unless the function has been interrupted under your own control by a signal or callback.

SEE ALSO

`line_get(), line_input_available()`

line_input_available

NAME

`line_input_available` - check if input on line

SYNOPSIS (C)

```
boolean GLAPI line_input_available(void);
```

SYNOPSIS (PERL)

```
$available = Glapi::Input_available();
```

DESCRIPTION

This function can be used by the application to check if text or an error message is available from the line module for processing. The use of this call before doing a call to `line_get()`, will avoid the application being suspended to wait until input is available. For two way simultaneous sessions it allows the application to check for error messages from the network even though the host application never normally replies. The available text or message must be read by a call to `line_get()`.

The application may call `line_input_available()` to poll for input and then do a `line_get()`. However, it is recommended to avoid polling and use either `line_wait()` or `line_select()` in stead.

RETURN VALUE

A value of 1 (true) is returned if there is a message waiting to be processed, otherwise a value of 0 (false) is returned.

line_connected

NAME

`line_connected` - check whether the line is up

SYNOPSIS (C)

```
boolean GLAPI line_connected(void);
```

SYNOPSIS (PERL)

```
$connected = Glapi::Connected();
```

DESCRIPTION

This function returns true or false depending on whether the line is up or not. The line is considered to be up after the connection is established and until the session is closed.

RETURN VALUE

The value of 1 (true) is returned if the line is up, 0 (false) is returned otherwise.

line_our_turn

NAME

`line_our_turn` - check if our turn

SYNOPSIS (C)

```
boolean GLAPI line_our_turn(void);
```

SYNOPSIS (PERL)

```
$ourTurn = Glapi::Our_turn();
```

DESCRIPTION

This function can be used to check if we have been given the turn. In two-way alternate (TWA) sessions, the host may not have finished sending us output, so we must wait for it to give us the turn. In most two-way simultaneous (TWS) sessions turn is sent, however in some it is not. This is why we only use this function with TWA to check if there is more input to come (not our turn) or if we are allowed to send something to the other side (our turn).

The `line_our_turn` status is only updated after a `line_get()`, `line_put()` or `line_write()` function has been called.

RETURN VALUE

A value of 1 (true) is returned if it is our turn, 0 (false) otherwise.

SEE ALSO

`line_simultaneous()`

line_simultaneous

NAME

`line_simultaneous` - check for two way simultaneous

SYNOPSIS (C)

```
boolean GLAPI line_simultaneous(void);
```

SYNOPSIS (PERL)

```
$simul = Glapi::Simultaneous();
```

DESCRIPTION

This function is used to check whether the session is two-way simultaneous (TWS) or not. Most DSA simultaneous sessions send turn, but some do not. This information is essential when flow control is concerned.

RETURN VALUE

A value of 1 (true) is returned if the session is two-way simultaneous (TWS), 0 (false) is returned if the session is two-way alternate (TWA).

SEE ALSO

`line_our_turn()`

line_demand_turn

NAME

`line_demand_turn` - demand the turn

SYNOPSIS (C)

```
void GLAPI line_demand_turn(void);
```

SYNOPSIS (PERL)

```
$rc = Glapi::Demand_turn();
```

DESCRIPTION

This function enables the application to get turn even if the host has it. The function guarantees the integrity of subsequent sends.

RETURN VALUE

No return value.

line_select

UNIX only

NAME

`line_select` - check line input

SYNOPSIS (C)

```
int GLAPI line_select (
        int    lid);
```

SYNOPSIS (PERL)

```
$fd = Glapi::Select($lid);
```

DESCRIPTION

This call may be used by an application that wishes to check the state of one of its lids. This call may be very useful if the application has started several line modules. Upon return from this call, the application will know if there is data available to be picked up by `line_get()` or receive a valid file descriptor that can be used in `poll()` or `select()` system calls to check input on the lid in question.

RETURN VALUE

The function will return a valid file descriptor if the lid is valid, but there is no data available. If the lid is not in use, it will return -1. If there is data available and a call to `line_get()` on the lid will return data, -2 is returned.

ERRORS

-1 is returned if the lid is not in use.

SEE ALSO

`line_get(), line_wait()`

line_waitcallback

NAME

`line_waitcallback` - sets a wait `CallBack` function

SYNOPSIS

Windows only

```
boolean GLAPI line_waitcallback (
        BOOL (WINAPI *CallBack)(void));
```

DESCRIPTION

This function replaces the default wait loop function in the `line_wait()` function. This allows applications to perform other tasks when Windows messages are received.

By default, the `CallBack` function is:

```
BOOL WINAPI WlistCallBackDefault(void)
{
    MSG     msg;

    if (PeekMessage (&msg, (HWND)NULL, 0, 0, PM_REMOVE)) {
        TranslateMessage (&msg);
        DispatchMessage (&msg);
    }
    return(TRUE);
}
```

If the `CallBack` function returns `FALSE`, the `line_wait()` function will terminate and return 99.

This function is called when a message has already been received, so a `PeekMessage()` or `GetMessage()` will always return `TRUE` immediately with the message. If running in a multi-thread environment, it is recommended to use `PeekMessage()`. The `WM_QUIT` message has already been tested before this `CallBack` function is called and has not been removed from the message queue. For applications needing a `PeekMessage()` type loop, then the `line_input_available()` function should be used rather than the `line_wait()` or `line_wait_lid()` functions.

RETURN VALUE

The return value is true if successful.

SEE ALSO

`line_wait(), line_wait_lid()`

line_init_keepalive

NAME

`line_init_keepalive` – initiate keep alive logic

SYNOPSIS (C)

```
void line_init_keepalive(int seconds);
```

SYNOPSIS (PERL)

```
Glapi::Init_keepalive($seconds);
```

DESCRIPTION

This function is used to initiate keep alive logic when the application is used to communicate through a Ggate system (e.g. when the OSI transport stack is running on another Host Links system). It sets the keepalive timeout interval to be used for that session. The timeout interval is sent in the 'logon packet' and informs Ggate about the maximum time that should elapse without any kind of traffic (data packets or keep alive packets) on that session. When the keepalive timer expires, the session is expected to be terminated by Ggate.

It is the GLAPI application's responsibility to send the keep alive packets (using the line_send_keepalive function) within the time interval set in this call.

This call must be issued **before** the line_start() call.

If the line_init_keepalive function is not called, GLAPI keep alive logic is not activated and only the keep alive timers configured for the TCP/IP stack will apply.

RETURN VALUE

No return value.

SEE ALSO

`line_send_keepalive()`

line_send_keepalive

NAME

`line_send_keepalive` – send keepalive packet

SYNOPSIS (C)

```
Void line_send_keepalive(void);
```

SYNOPSIS (PERL)

```
Glapi::Send_keepalive();
```

DESCRIPTION

This function is used to send a keep alive packet. It should be used before the time specified in the line_init_keeaplive function has expired if no other data packet has been sent on that session within that time interval. Failure to send the keep alive packet (or data) within the specified keepalive interval, will result in the session beeing aborted by Ggate on the other side.

RETURN VALUE

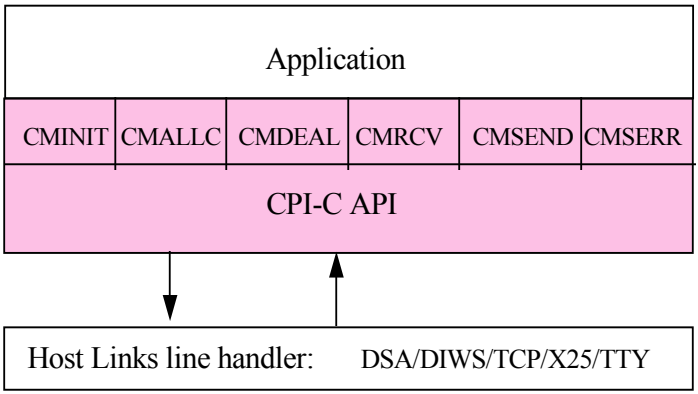
No return value.

SEE ALSO

`line_init_keepalive()`

CPI-C APIs

Architecture



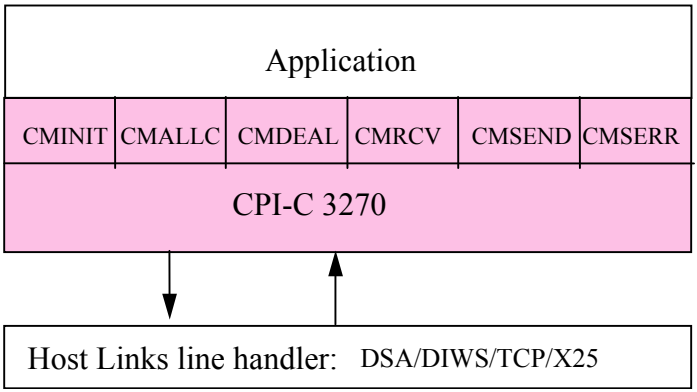
An application using the CPI-C API will at run-time connect to a Gline line handler module. When the Initialize_Conversation function is used, the configuration file, called `cpic.cfg`, is read and the configured communication line handler module will be started. The application and the line handler are separate executables.

The line handler can be one of the following:

Line handler	Communication type	Comms. API
`gl_dsa`	DSA or DSA/ISO WorkStation (DIWS) session over OSI transport	XTI, TLI
	DSA or DSA/ISO WorkStation (DIWS) session over RFC1006 (G&R OSI TP0 over TCP/IP)	Sockets
`gl_tcp`	'Raw' TCP/IP	Sockets
	Telnet	Sockets
	TNVIP client	Sockets
	TN3270 client	Sockets
	TN5250 client	Sockets
`gl_x25`	PAD	Provider specific
	'Raw' X25	Provider specific
`gl_tty`	Asynchronous line	Basic OS

The line handler is chosen by giving one of the following parameters to Gline: `-LI DSA`, `-LI DIWS`, `-LI TCP`, `-LI X25` or `-LI TTY`.

CPI-C 3270



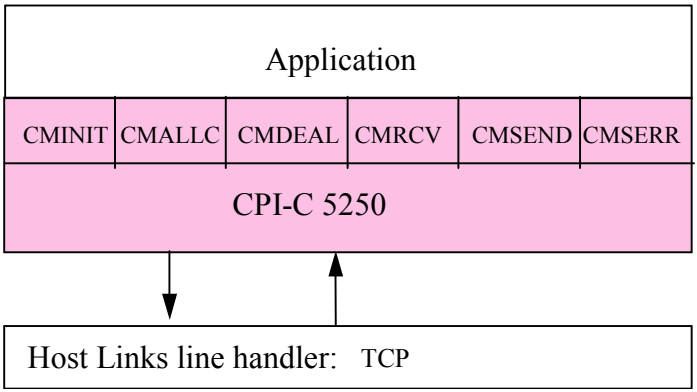
An application using the *CPI-C 3270 API* will at run-time connect to a Gline line handler module. When the `Initialize_Conversation` function is used, the configuration file, called `cpic.cfg`, is read and the configured communication line handler module will be started. The application and the line handler are separate executables.

The line handler can be one of the following:

Line handler	Communication type	Comms. API
`gl_dsa`	DSA or DSA/ISO WorkStation (DIWS) session over OSI transport via OSF/Janus in Bull frontend.	XTI, TLI
	DSA or DSA/ISO WorkStation (DIWS) session over RFC1006 (G&R OSI TP0 over TCP/IP) via OSF/Janus in Bull frontend.	Sockets
`gl_tcp`	TN3270 client	Sockets
`gl_x25`	'Raw' X25	Provider specific

The line handler is chosen by giving one of the following parameters to Gline: `-LI DSA`, `-LI DIWS`, `-LI TCP` or `-LI X25`.

CPI-C 5250



An application using the *CPI-C 5250 API* will at run-time connect to a Gline line handler module. When the `Initialize_Conversation` function is used, the configuration file, called `cpic.cfg`, is read and the configured communication line handler module will be started. The application and the line handler are separate executables.

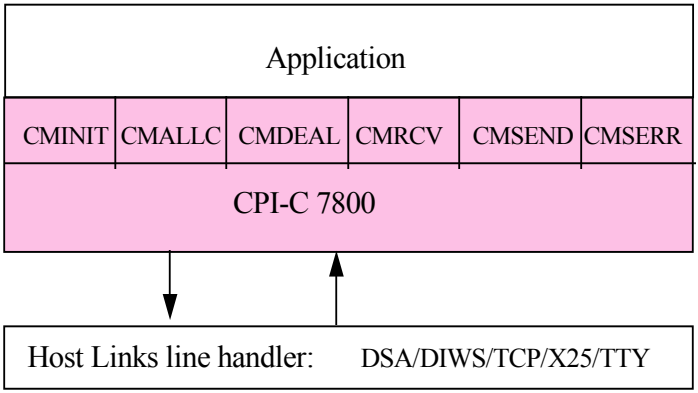
The line handler can be one of the following:

Line handler	Communication type	Comms. API
`gl_tcp`	TN5250 client	Sockets

The line handler is chosen by giving the following parameters to Gline:

```
-LI TCP.
```

CPI-C 7800

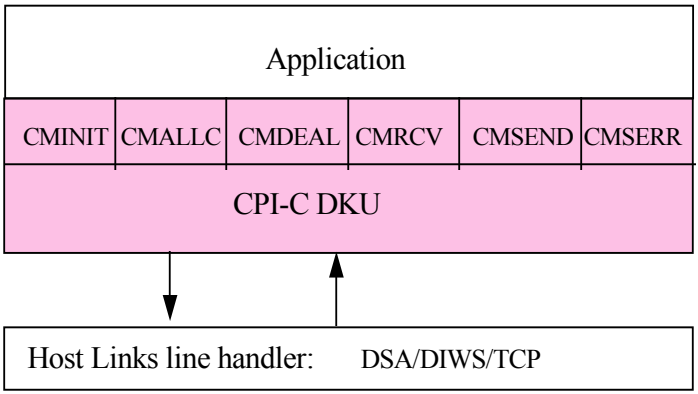


An application using the *CPI-C 7800 API* will at run-time connect to a Gline line handler module. When the Initialize_Conversation function is used, the configuration file, called `cpic.cfg`, is read and the configured communication line handler module will be started. The application and the line handler are separate executables. The line handler can be one of the following:

Line handler	Communication type	Comms. API
`gl_dsa`	DSA or DSA/ISO WorkStation (DIWS) session over OSI transport	XTI, TLI
	DSA or DSA/ISO WorkStation (DIWS) session over RFC1006 (G&R OSI TP0 over TCP/IP)	Sockets
`gl_tcp`	'Raw' TCP/IP	Sockets
	Telnet	Sockets
	TNVIP client	Sockets
`gl_x25`	PAD	Provider specific
	'Raw' X25	Provider specific
`gl_tty`	Asynchronous line	Basic OS

The line handler is chosen by giving one of the following parameters to Gline: `-LI DSA`, `-LI DIWS`, `-LI TCP`, `-LI X25` or `-LI TTY`.

CPI-C DKU



An application using the *CPI-C DKU API* will at run-time connect to a Gline line handler module. When the Initialize_Conversation function is used, the configuration file, called `cpic.cfg`, is read and the configured communication line handler module will be started. The application and the line handler are separate executables.

The line handler can be one of the following:

Line handler	Communication type	Comms. API
`gl_dsa`	DSA or DSA/ISO WorkStation (DIWS) session over OSI transport	XTI, TLI
	DSA or DSA/ISO WorkStation (DIWS) session over RFC1006 (G&R OSI TP0 over TCP/IP)	Sockets
`gl_tcp`	TNVIP client	Sockets

The line handler is chosen by one of the following parameters: `-LI DSA`, `-LI DIWS` or `-LI TCP`.

CPI-C compatibility

G&R/CPI-C API is compatible with the X/Open CPI-C version 2. The `cpic.h` file supplied with the *G&R/GlAPI SDK* also contains version 1 compatible functions. The version 1 function uses uppercase function names and are supplied in the UNIX G&R/CPI-C static link libraries for compatibility with previous versions of CPI-C object files, however, only the following functions are supplied in the libraries; CMALLC, CMDEAL, CMINIT, CMRCV, CMSEND and CMSERR. These functions map directly to their corresponding version 2 functions with the exception that they return the `return_code` value. For future compatibility, we do not recommend using there uppercase function, and any version 1 CPI-C source files should be converted to the version 2 lowercase functions.

There are some functional differences with the G&R CPI-C API previous to 5.0. These differences are in the initial states of the CPI-C concerning the Deallocation type and the Send type:

`Deallocate type`	pre-5.0: `CM_DEALLOCATE_ABEND` from 5.0: `CM_DEALLOCATE_SYNC_LEVEL` See `Set_Deallocate_Type` on page 94 for details
`Send type`	pre-5.0: `CM_SEND_AND_PREP_TO_RECEIVE` from 5.0: `CM_BUFFER_DATA` See `Set_Send_Type` on page 104 for details

Also, the `Receive` function in version 1 only returned when it received the turn from the host and therefore always returned the `CM_SEND_RECEIVED` status. Now this function returns when any message is received from the host and only returns the `CM_SEND_RECEIVED` status when it has received the turn from the host. This allows the CPI-C application to send multiple messages from the host in smaller buffers.

The cpic.cfg option `-SS` is supplied to change the CPI-C states and Receive function to work as in the pre-5.0 versions.

Gallagher & Robertson has included some private extensions to the CPIC interface. This is clearly indicated where the extension is documented.

CPI-C States & State-transitions

During the execution phase, the local application changes its state each time a CPI-C call is issued. Each CPI-C call causes the local application to change from a certain state A to another state B. If the local application is not in the proper state when the CPI-C call is made, a return code indicates the error that occurred. If an error is detected, the state transition does not occur.

A *CPI-C API* operates in 4 states: RESET, INIT, SEND and RECEIVE. It always starts off in RESET state. It will be set to INIT state after a successful return from the `Initialize_Conversation` call.

In INIT state, a successful connection to the remote host application, with a call to `Allocate`, will by default put the local application in SEND state. However by setting the parameter `-SND OFF`, the local application will be in RECEIVE state after a successful call to `Allocate`.

In RECEIVE state, the local application may issue calls to `Receive` or `Deallocate`. `Receive` will put it in SEND state and `Deallocate` will put it in RESET state.

In SEND state, the local application may issue calls to `Send_Data` or `Deallocate`. `Send_Data` will put it in RECEIVE state and `Deallocate` will put it in RESET state.

A call to `Send_Error` will put the local application in RECEIVE state, both when issued in SEND and RECEIVE state.

CPI-C API variables and definitions

SIDEINFO structure

```
typedef struct side_info_entry {
unsigned char                  sym_dest_name[8];
unsigned char                  partner_LU_name[17];
unsigned char                  reserved[3];
XC_TP_NAME_TYPE                TP_name_type;
unsigned char                  TP_name[64];
unsigned char                  mode_name[8];
CM_CONVERSATION_SECURITY_TYPE conversation_security_type;
unsigned char                  security_user_ID[8];
unsigned char                  security_password[8];
} SIDE_INFO;
```

Sym_dest_name	Symbolic destination name.
Partner_LU_name	Fully Q'd PLU name.
Reserved	Reserved.
TP_name_type	Set to: XC_APPLICATION_TP XC_SNA_SERVICE_TP
TP_name	TP name
Mode_name	Mode name
Conversation_security_type	Set to: CM_SECURITY_NONE CM_SECURITY_SAME CM_SECURITY_PROGRAM
Security_user_ID	User_ID
Security_password	Password

CPIC_FIELD_INFO structure

G&R private extension

```
typedef struct CPIC_FIELD_INFO {
   int fldindex;
   int startpos;
   int endpos;
} CPIC_FIELD_INFO;
```

fldindex	If zero or higher: The field index of the current field. A value of -1 indicates that the rest of the members of this structure are void
startpos	Offset from the beginning of the screen buffer to where the field starts.
endpos	Offset from the beginning of the screen buffer to where the field ends.

definitions

```
#include <cpic.h>
```

Windows:

```
#define windows
#define windows16
#define windows32
#include <wincpic.h>
```

Unix:

```
#define unix
```

Perl:

```
use Cpic;          # For the non-emulating API
use Cpic7800;      # For CPI-C 7800 API
use Cpic3270;      # For CPI-C IBM 3270 API
use Cpic5250;      # For CPI-C IBM 5250 API
use CpicDku;       # For CPI-C DKU API
```

CPI-C API functions list

Function	Description
Accept_Conversation	Accept a conversation
Allocate	Connect to remote application
Deallocate	close connection and stop the line handler.
Initialize_Conversation	read parameters and start line handler
Prepare_To_Receive	prepare to receive
Receive	return next form to the application
Request_To_Send	request to send
Send_Data	read "keyboard input" and send
Send_Error	send BREAK to the host application
Set_Conversation_Type	set conversation type
Set_Deallocate_Type	set how the conversation will be deallocated
Set_Mode_Name	set mode name
Set_Partner_LU_Name	set partner LU name
Set_Prepare_To_Receive_Type	set prepare to receive type
Set_Receive_Type	set the conversations receive type
Set_Send_Type	set the conversations send type
Set_Sync_Level	set the sync level
Set_TP_Name	set the remote program name
Set_Conversation_Security_Type	set the conversation security type (same as xcscst)
Set_Conversation_Security_User_ID	set the conversations security id

Function	Description
`Set_Conversation_Security_Password`	set the conversations security password id
`Set_CPIC_Side_Information`	set side information
`Set_Conversation_Security_Type`	set the conversation security type (same as `cmscst`)
`WinCPICIsBlocking`	test if blocking
`WinCPICSetBlockingHook`	set blocking hook
`WinCPICUnhookBlockingHook`	removes blocking hook
`WinCPICStartup`	initialization call
`WinCPICCleanup`	termination call
`char * api_msg`	deliver a message based on the CPI-C retcode

CPI-C Emulation API functions list

Function	Description
`Allocate`	connect to remote application
`Deallocate`	close connection and stop the line handler.
`Initialize_Conversation`	read parameters and start line handler
`Receive`	return next form to the application
`Send_Data`	read "keyboard input" and send
`Send_Error`	send BREAK to the host application
`char * api_msg`	deliver a message based on the CPI-C retcode
`Get_Field_Info`	Retrieve information regarding variable fields

CPI-C functions

Accept_Conversation (cmaccp)

not emulation
CPI-C

NAME

`Accept_Conversation (cmaccp)` - accept a conversation

SYNOPSIS (C)

```
#include "cpic.h"
/* Accept_Conversation */
CM_ENTRY cmaccp(
        unsigned char CM_PTR conversation_id,
        CM_INT32 CM_PTR      return_code);
```

SYNOPSIS (PERL)

```
use Cpic;
($cid, $rc) = Cpic::Accept_Conversation();
```

DESCRIPTION

The `Accept_Conversation (cmaccp)` call accepts an incoming conversation. Like `Initialize_Conversation`, this call initializes values for various conversation characteristics.

By default the line handle parameters will be picked up from the `Default` section in the `cpic.cfg` file, however, a specific node name may be supplied in the default `sym_dest_name` by using the `Set_CPIC_Side_Infarmation` function. When this function is called, it will only return when an incoming connection has been received. The `-LW nn` option can be used to specify a timeout, in which case `CM_ALLOCATE_FAILURE_NO_RETRY` is returned.

The `conversation_id` output parameter specifies the conversation identifier assigned to the conversation. CPI-C supplies and maintains the

`conversation_id`. When the `return_code` is set equal to `CM_OK`, the value returned in this parameter is used by the program on all subsequent calls issued for this conversation.

RETURN VALUE

`return_code`	`CM_OK`
	The application/API has received a successful connection from the host application. It is now in SEND state and a call to `Send_Data` should be made to read the first form.
	`CM_PROGRAM_PARAMETER_CHECK`.
	An invalid value was specified in the CPI-C config file, or Side Information. API is in RESET state.
	`CM_ALLOCATE_FAILURE_NO_RETRY`
	The incoming connection was aborted, See the `-LW` parameter.

ERRORS

If an error has occurred, `return_code` will be different from `CM_OK` and the value will describe the error.

Allocate (cmallc)

NAME

`Allocate (cmallc)` - connect to remote application

SYNOPSIS (C)

```
#include "cpic.h"

/* Allocate */
CM_ENTRY cmallc (
        unsigned char CM_PTR conversation_id,
        CM_INT32 CM_PTR      return_code);
```

SYNOPSIS (PERL)

```
use CpicXXXX;
$rc = CpicXXXX::Allocate($cid);
```

DESCRIPTION

When `Allocate (cmallc)` is called, a connection to the configured host application will be established, and the local application will be put in SEND state (default case). In this state if the host takes the initiative and sends a (log-on) form it will be purged and never delivered to the local application. However by supplying the parameter `-SND OFF`, the local application will be in RECEIVE state after a successful call to `Allocate` and the first form can be received in a following call to `Receive`.

In order to handle the situations where the host application does not send a log-on screen or welcome-banner and/or a Two-Way-Simultaneous (Full duplex) connection has been negotiated, we have introduced a parameter, `-CW OFF`, that will force an immediate return to the application when the connection is established.

An example of a typical connection to a host would be:

```
#ifdef windows
WORD    wVer = 2;
WCPICDATA       CPICData;

    if (WinCPICStartup(wVer, &CPICData) != 0)
        return (-1);
#endif
```

```
Initialize_Conversation(conv_id, "tssdps8", &retcode);
if (retcode) {
    send_disconnect();
    return (-1);
}

Allocate(conv_id, &retcode);
if (retcode) {
    send_disconnect();
    return (-1);
}
/* set send type to send with turn */
sndtyp = CM_SEND_AND_PREP_TO_RECEIVE;
Set_Send_Type(conv_id, &sndtyp, &retcode);
```

RETURN VALUE

`return_code`	`CM_OK`
	The application/API has made a successful connection to the host application. It is now in RECEIVE state and a call to `Receive` should be made to read the first form.
	`CM_PROGRAM_STATE_CHECK`
	The application/API was in SEND or RECEIVE state.
	`CM_PROGRAM_PARAMETER_CHECK`
	The conversation_id is undefined or the application/ API is in RESET state.
	`CM_ALLOCATE_FAILURE_NO_RETRY`
	The API was not able to connect to the host application. Check connection parameters in the configuration file, and verify that the host application is running. It might be a good idea to try to make a similar connection from *`G3270`*, *`G5250`*, *`V78sim`*, or *`Qsim`*.

ERRORS

If an error has occurred, `return_code` will be different from `CM_OK` and the value will describe the error.

Deallocate (cmdeal)

NAME

Deallocate (cmdeal) - close connection and stop the line handler.

SYNOPSIS (C)

```
#include "cpic.h"

CM_ENTRY cmdeal (
        unsigned char CM_PTR conversation_id,
        CM_INT32 CM_PTR      return_code);
```

SYNOPSIS (PERL)

```
use CpicXXXX;
$rc = CpicXXXX::Deallocate($cid);
```

DESCRIPTION

When Deallocate (cmdeal) is called, the connection is released. Whether the connection to the host application is closed and the *Host Links* line handler is terminated depends on the Deallocate type which can be changed with Set_Deallocate_Type. By default the connection is returned to a pool and may used again when the same symbolic name is requested, however normal usage to disconnect from the host would be;

```
    /* set deallocate type to be CM_DEALLOCATE_ABEND */
    /* to force disconnect */
    dealtyp = CM_DEALLOCATE;
    Set_Deallocate_Type(conv_id, &dealtyp, &retcode);
    Deallocate(conv_id, &retcode);
#ifdef windows
    WinCPICCleanup();
#endif
```

To reconnect, the application must make new calls to Initialize_Conversation and Allocate.

RETURN VALUE

`return_code`	`CM_OK`
	The application/API has received a new form from the host application, and it is now in SEND state. The received form is returned in buffer, the received length is always a full screen, 1920 characters.
	`CM_PROGRAM_PARAMETER_CHECK`
	The conversation_id is undefined or the application/ API is in RESET state.

ERRORS

If an error has occurred, `return_code` will be different from `CM_OK` and the value will describe the error.

Get_Field_Info (cmfld)

Emulation
CPI-C only

G&R private extension

NAME

`Get_Field_Info (cmfld)` - retrieve variable field information

SYNOPSIS (C)

```
#include "cpic.h"

/* Get_Field_Info */
CM_ENTRY cmfld(
        unsigned char CM_PTR conversation_id,
        CPIC_FIELD_INFO CM_PTR field_info,
        CM_INT32 CM_PTR field_struct_size,
        CM_INT32 CM_PTR      return_code);
```

SYNOPSIS (PERL)

```
use CpicXXXX;
($fieldnum,$start,$endpos,$rc) =
   CpicXXXX::Get_Field_Info($cid,$fieldnum);
```

DESCRIPTION

`Get_Field_Info (cmfld)` is used by a program to pick up information about the variable fields (if any) present on the current screen (the screen last retrieved by the `cmrcv()` function call). When calling this function, supply as a parameter the number of the field (starting with 1) you wish to retrieve information about. To retrieve information on about all fields, start with 1 and increase by 1 until the returned field number is -1.

This function is a Gallagher & Robertson private extension to the CPIC interface.

RETURN VALUE

`return_code`	`CM_OK` `CM_PROGRAM_PARAMETER_CHECK`. The conversation_id is undefined or the application/API is in RESET state.

ERRORS

If an error has occurred, `return_code` will be different from `CM_OK` and the value will describe the error. The returned field number will contain a value of `-1` when you have requested information about a non-existing variable field.

Initialize_Conversation (cminit)

NAME

`Initialize_Conversation (cminit)` - read parameters and start line handler

SYNOPSIS (C)

```
#include "cpic.h"

CM_ENTRY cminit (
        unsigned char CM_PTR conversation_id,
        unsigned char CM_PTR sym_dest_name,
        CM_INT32 CM_PTR      return_code);
```

SYNOPSIS (PERL)

```
use CpicXXXX;
($cid, $rc) = CpicXXXX::Initialize_Conversation($name);
```

DESCRIPTION

When `Initialize_Conversation (cminit)` is called, the configuration file, `cpic.cfg`, is read and the selected Gline line handler is started. The host label given to `Initialize_Conversation` in `sym_dest_name` is used by `Initialize_Conversation` to find connection parameters for the host application. After reading the configuration file, but before starting the line handler, `Initialize_Conversation` will check the following environment variables and pass their content as Gline parameter values:

Environment variables	Line handler parameters
INITIAL_COR	-DA
REMNODE	-DN
REMMB_EXT	-DX
LOCNODE	-LN
LOCMB	-MN
LOCMB_EXT	-MX

Environment variables	Line handler parameters
USERINFO	`-UR`
SECURITY	`-PW`
EMU_MODEL	`-TM`

If an environmental variable has been set, it will override any value specified for that parameter in the configuration file. For more information on the line handlers and their parameters, see the *`Gline`* manual.

INITIAL STATES

`deallocate_type`	`CM_DEALLOCATE_SYNC_LEVEL`
`send_type`	`CM_BUFFER_DATA`
`receive type`	`CM_RECEIVE_AND_WAIT`

RETURN VALUE

`conversation_id`	A unique conversation_id will be returned by `Initialize_Conversation` when it completes its operation without error.

`return_code`	`CM_OK`
	Configuration file has been analyzed and the configured line handler has been started. The API is now in INIT state.
	`CM_PROGRAM_STATE_CHECK`
	The application/API was not in RESET state.
	`CM_PROGRAM_PARAMETER_CHECK`
	An error was found in the configuration file, or it was not possible to start the configured line handler.

ERRORS

If an error has occurred, `return_code` will be different from `CM_OK` and the value will describe the error.

Prepare_To_Receive (cmptr)

not emulation
CPI-C

NAME

`Prepare_To_Receive (cmptr)` - prepare to receive

SYNOPSIS (C)

```
#include "cpic.h"

/* Prepare_To_Receive */
CM_ENTRY cmptr(
        unsigned char CM_PTR conversation_id,
        CM_INT32 CM_PTR      return_code);
```

SYNOPSIS (PERL)

```
use Cpic;
$rc = Cpic::Prepare_To_Receive($cid);
```

DESCRIPTION

`Prepare_To_Receive (cmptr)` — change a conversation from Send to Receive state in preparation to receive data.

This function currently has no effect and is supplied for program compatibility reasons.

RETURN VALUE

`return_code`	`CM_OK` `CM_PROGRAM_PARAMETER_CHECK`.

ERRORS

If an error has occurred, `return_code` will be different from `CM_OK` and the value will describe the error.

Receive (cmrcv)

NAME

`Receive (cmrcv)` - return next form to the application

SYNOPSIS (C)

```
#include "cpic.h"

CM_ENTRY cmrcv (
        unsigned char CM_PTR conversation_id,
        unsigned char CM_PTR buffer,
        CM_INT32 CM_PTR      request_length,
        CM_INT32 CM_PTR      data_received,
        CM_INT32 CM_PTR      received_length,
        CM_INT32 CM_PTR      status_received,
        CM_INT32 CM_PTR      request_to_send_received,
        CM_INT32 CM_PTR      return_code);
```

SYNOPSIS (PERL)

```
use CpicXXXX;
($buffer,$dr,$status,$rts,$rc) = CpicXXXX::Receive($cid);
```

DESCRIPTION

The `Receive (cmrcv)` function is slightly different in the CPI-C line handler API and the CPI-C emulation API. The application must be in RECEIVE state before it is allowed to receive data.

In the line handle CPI-C API, `Receive` will wait until the host application has sent some data. The `status_received` will indicate whether the turn was received from the host or not. If the `-SS ON` option is set, then it will wait for the turn and concatenate host data into the same receive buffer. If the `-SS ON` option is set, then CPI-C will Deallocate on reception of the turn. This option should not normally be used.

The emulation APIs will wait until the host has finished sending the next form. As default a cleared screen is not considered as a form in the CPI-C 3270, 5250, 7800 and DKU APIs. However, when the parameter `-BS ON` is specified, the `Receive` function in CPI-C 3270, 5250, 7800 and DKU API will return also when the screen is "empty" (cleared). When the form is received and the "terminal" is allowed to send again, `Receive` will return a buffer containing the current screen/form. Since the emulation APIs (CPI-C 3270, 5250, 7800 and DKU) always deliver a copy of the screen (i.e 24*80 characters), the size of the returned buffer will always be 1920 characters. In the CPI-C API, the size of the buffer will vary as this interface delivers the data as it is received.

In the CPI-C APIs we have added a parameter, `-RW NN`, that allows you to specify a maximum wait time in a `Receive` call. The default is to wait indefinitely for the host application to respond, but when a number is specified with the `-RW` parameter, `Receive` will close the connection and return to the application in RESET state when the configured number of seconds have elapsed. In this case the return code, `CM_RETCODE`, will be `CM_DEALLOCATED_ABEND_TIMER`.

Typically your application would continue to call the receive function until is received the turn indicated by the `CM_SEND_RECEIVED` status:

```
retcode = CM_OK;
status = CM_NO_STATUS_RECEIVED;
while (retcode == CM_OK && status == CM_NO_STATUS_RECEIVED) {
    Receive(conv_id, inpbuf, &inplen, &datarec, &reclen,
          &status, &rts, &retcode);
    if (retcode == CM_OK)
        treat_buffer(inpbuf, reclen);
}
```

See also the `-SS, -DG` options for the `cpic.cfg` file.

RETURN VALUE

Parameter	Description
buffer	Address of buffer where the received form will be returned.
data_received	Always CM_COMPLETE_DATA_RECEIVED, which means that the complete form has been received.
received_length	Always same size as the screen buffer in the CPI-C 3270, 5250, 7800 and DKU APIs, but will vary in the basic CPI-C API, and reflect the actual number of bytes received from the host. The 3270, 5250 and DKU APIs only support 24*80 screens at the moment, so for these APIs the value will always be 1920 (characters).
status_received	Always CM_SEND_RECEIVED, which means that the host application is ready to receive data from us.
request_to_send_received	Always CM_REQ_TO_SEND_NOT_RECEIVED because we do not expect such a notification.
return_code	CM_OK The application/API has received a new form from the host application, and it is now in SEND state. The received 3270, 5250, 7800 or DKU form is returned in buffer, the received length is always a full screen (3270/5250/DKU: 1920 characters). CM_PROGRAM_STATE_CHECK The application/API was in INIT state. CM_PROGRAM_PARAMETER_CHECK The conversation_id is undefined or the application/API is in RESET state. CM_DEALLOCATED_NORMAL The status_received is CM_SEND_RECEIVED, the turn was received,

Parameter	Description
	and the -DG ON option is used. CM_DEALLOCATED_ABEND The connection to the host application has been closed. It can be a result of a previous command (sent by Send_Data) that caused the host application to terminate, or it might be an abnormal termination. CM_DEALLOCATED_ABEND_TIMER The connection to the host application has been closed due to an internal time-out. The host application did not respond within the wait time configured with the -RW nn parameter, and the API has now closed the connection, terminated the line handler and returned to RESET state. CM_UNSUCCESSFUL This code will be returned is the Receive type was CM_RECEIVE_IMMEDIATE and there was not data available

ERRORS

If an error has occurred, return_code will be different from CM_OK and the value will describe the error.

Request_To_Send (cmrts)

not emulation CPI-C

NAME

`Request_To_Send (cmrts)` - request to send

SYNOPSIS (C)

```
#include "cpic.h"

/* Request_To_Send */
CM_ENTRY cmrts(
        unsigned char CM_PTR conversation_id,
        CM_INT32 CM_PTR      return_code);
```

SYNOPSIS (PERL)

```
use Cpic;
$rc = Cpic::Request_To_Send($cid);
```

DESCRIPTION

The local program uses the `Request_To_Send (cmrts)` call to notify the remote program that the local program would like to enter Send state for a given conversation.

This function currently has no effect and is supplied for program compatibility reasons.

RETURN VALUE

`return_code`	`CM_OK`
	`CM_PROGRAM_PARAMETER_CHECK.`
	The conversation_id is undefined or the application/API is in RESET state.

ERRORS

If an error has occurred, `return_code` will be different from `CM_OK` and the value will describe the error.

Send_Data (cmsend)

NAME

`Send_Data (cmsend)` - read "keyboard input" and send

SYNOPSIS (C)

```
#include "cpic.h"

CM_ENTRY cmsend (
        unsigned char CM_PTR conversation_id,
        unsigned char CM_PTR buffer,
        CM_INT32 CM_PTR      send_length,
        CM_INT32 CM_PTR      request_to_send_received,
        CM_INT32 CM_PTR      return_code);
```

SYNOPSIS (PERL)

```
use CpicXXXX;
$rc = CpicXXXX::Send($cid,$buffer,$length,$rsr);
```

DESCRIPTION

When the application is in SEND state, it is allowed to send data to the host application. With the CPI-C API, the buffer is sent to the application as delivered in the `Receive` call. This mean that the application is responsible for sending data in a format accepted by the host application. With the CPI-C 3270, 5250, 7800 and DKU API the buffer is filled with simulated keyboard input, which will be used to fill in the variable fields of the current form and sent to the host application. A list of supported function keys and their codes are presented in the *CPI-C 3270 API: keyboard input*, *5250 API: keyboard input*, *CPI-C 7800 API: keyboard input* and *CPI-C DKU API: keyboard input* chapters.

To send a message with the turn to the host so as to be in RECEIVE state after the `Send_Data` would be:

```
sndtyp = CM_SEND_AND_PREP_TO_RECEIVE;
Set_Send_Type(conv_id, &sndtyp, &retcode);
Send_Data(conv_id, outbuf, &sndlen, &rts, &retcode);
```

To send a message without the turn to the host so as to remain in SEND state after the `Send_Data` would be:

```
sndtyp = CM_BUFFER_DATA;
Set_Send_Type(conv_id, &sndtyp, &retcode);
Send_Data(conv_id, outbuf, &sndlen, &rts, &retcode);
```

In the above example, the data would not be sent to the host until the line handler buffer was full (see the Glink `-LL` parameter) or the `CM_SEND_AND_PREP_TO_RECEIVE` send state was set.

RETURN VALUE

`return_code`	`CM_OK`
	The application/API has sent its input to the host application, and it is now in RECEIVE state.
	`CM_PROGRAM_STATE_CHECK`
	The application/API was in INIT state.
	`CM_PROGRAM_PARAMETER_CHECK`
	The conversation_id is undefined or the application/API is in RESET state.
	`CM_DEALLOCATED_ABEND`
	The connection to the host application has unexpectedly been closed or aborted, most likely be the host application. Check the log or trace to find the reason.

ERRORS

If an error has occurred, `return_code` will be different from `CM_OK` and the value will describe the error.

Send_Error (cmserr)

NAME

`Send_Error (cmserr)` - send BREAK to GCOS host

SYNOPSIS (C)

```
#include "cpic.h"

CM_ENTRY cmserr (
        unsigned char CM_PTR conversation_id,
        CM_INT32 CM_PTR      request_to_send_received,
        CM_INT32 CM_PTR      return_code);
```

SYNOPSIS (PERL)

```
use CpicXXXX;
$rc = CpicXXXX::Send_Error($cid,$rsr);
```

DESCRIPTION

This function is used by the application to send a BREAK to the host. The BREAK is sent without regard to the current state, and the local application will be in RECEIVE state when the function returns. The behavior of the host is different from one host subsystem to another.

RETURN VALUE

`return_code`	`CM_OK`
	The application/API has sent a BREAK to the host application, and it is now in RECEIVE state.
	`CM_PROGRAM_STATE_CHECK`
	The application/API was in INIT state.
	`CM_PROGRAM_PARAMETER_CHECK`
	The conversation_id is undefined or the application/API is in RESET state.
	`CM_DEALLOCATED_ABEND`
	The connection to the host application has been closed or aborted, most likely by the host application. Check the log or trace to find the reason. The application/API will now be in RESET state.

ERRORS

If an error has occurred, `return_code` will be different from `CM_OK` and the value will describe the error.

Set_Conversation_Type (cmsct)

not emulation
CPI-C

NAME

`Set_Conversation_Type (cmsct)` - set conversation type

SYNOPSIS (C)

```
#include "cpic.h"

/* Set_Conversation_Type */
CM_ENTRY cmsct(
        unsigned char CM_PTR conversation_id,
        CM_INT32 CM_PTR      conv_type,
        CM_INT32 CM_PTR      return_code);
```

SYNOPSIS (PERL)

```
use Cpic;
$rc = Cpic::Set_Conversation_Type($cid, $ctype);
```

DESCRIPTION

`Set_Conversation_Type (cmsct)` is used by a program to set the `conversation_type` characteristic for a given conversation. It overrides the value that was assigned when the `Initialize_Conversation` call was issued.

This function currently has no effect and is supplied for program compatibility reasons.

RETURN VALUE

`return_code`	`CM_OK` `CM_PROGRAM_PARAMETER_CHECK`. The conversation_id is undefined or the application/API is in RESET state.

ERRORS

If an error has occurred, `return_code` will be different from `CM_OK` and the value will describe the error.

Set_Deallocate_Type (cmsdt)

not emulation CPI-C

NAME

`Set_Deallocate_Type (cmsdt)` - set how the conversation will be deallocated

SYNOPSIS (C)

```
#include "cpic.h"

/* Set_Deallocate_Type */
CM_ENTRY cmsdt(
        unsigned char CM_PTR conversation_id,
        CM_INT32 CM_PTR      deallocate_type,
        CM_INT32 CM_PTR      return_code);
```

SYNOPSIS (PERL)

```
use Cpic;
$rc = Cpic::Set_Deallocate_Type($cid, $dtype);
```

DESCRIPTION

`Set_Deallocate_Type (cmsdt)` is used by a program to set the `deallocate_type` characteristic for a given conversation. `Set_Deallocate_Type` overrides the value that was assigned when the `Initialize_Conversation` or `Accept_Conversation` call was issued. See also the `-SS, -DD,` options for the `cpic.cfg` file.

The possible values for the `deallocate_type` are:

`CM_DEALLOCATE_SYNC_LEVEL`	Releases the conversation to the pool and the connection to the host is not disconnected.

CM_DEALLOCATE_ABEND	Releases the conversation and the connection to the host is disconnected and the line handler terminated

RETURN VALUE

return_code	CM_OK CM_PROGRAM_PARAMETER_CHECK. The conversation_id is undefined or the application/API is in RESET state.

ERRORS

If an error has occurred, return_code will be different from CM_OK and the value will describe the error.

Set_Mode_Name (cmsmn)

not emulation CPI-C

NAME

`Set_Mode_Name (cmsmn)` - set mode name

SYNOPSIS (C)

```
#include "cpic.h"

/* Set_Mode_Name */
CM_ENTRY cmsmn(
        unsigned char CM_PTR  conversation_id,
        unsigned char CM_PTR mode_name,
        CM_INT32 CM_PTR      mode_name_len,
        CM_INT32 CM_PTR      return_code);
```

SYNOPSIS (PERL)

```
N/A
```

DESCRIPTION

`Set_Mode_Name (cmsmn)` is used by a program to set the `mode_name` and `mode_name_length` characteristics for a conversation. `Set_Mode_Name` overrides the current values that were originally acquired from the side information using the `sym_dest_name`.

This function currently has no effect and is supplied for program compatibility reasons.

RETURN VALUE

`return_code`	`CM_OK` `CM_PROGRAM_PARAMETER_CHECK`. The conversation_id is undefined or the application/API is in RESET state.

ERRORS

If an error has occurred, `return_code` will be different from `CM_OK` and the value will describe the error.

Set_Partner_LU_Name (cmspln)

not emulation
CPI-C

NAME

`Set_Partner_LU_Name (cmspln)` - set partner LU name

SYNOPSIS (C)

```
#include "cpic.h"

/* Set_Partner_LU_Name */
CM_ENTRY cmspln(
        unsigned char CM_PTR conversation_id,
        unsigned char CM_PTR partner_LU_name,
        CM_INT32 CM_PTR      partner_LU_name_len,
        CM_INT32 CM_PTR      return_code);
```

SYNOPSIS (PERL)

```
N/A
```

DESCRIPTION

`Set_Partner_LU_Name (cmspln)` is used by a program to set the `partner_LU_name` and `partner_LU_name_length` characteristics for a conversation. `Set_Partner_LU_Name` overrides the current values that were originally acquired from the side information using the `sym_dest_name`.

This function currently has no effect and is supplied for program compatibility reasons.

RETURN VALUE

`return_code`	`CM_OK` `CM_PROGRAM_PARAMETER_CHECK`. The conversation_id is undefined or the application/API is in RESET state.

ERRORS

If an error has occurred, `return_code` will be different from `CM_OK` and the value will describe the error.

Set_Prepare_To_Receive_Type (cmsptr)

not emulation
CPI-C

NAME

`Set_Prepare_To_Receive_Type (cmsptr)` - set prepare to receive type

SYNOPSIS (C)

```
#include "cpic.h"

/* Set_Prepare_To_Receive_Type */
CM_ENTRY cmsptr(
        unsigned char CM_PTR conversation_id,
        CM_INT32 CM_PTR      prep_to_rec_type,
        CM_INT32 CM_PTR      return_code);
```

SYNOPSIS (PERL)

```
use Cpic;
$rc = Cpic::Set_Prepare_To_Receive_Type($cid,$ptype);
```

DESCRIPTION

`Set_Prepare_To_Receive_Type (cmsptr)` is used by a program to set the `prepare_to_receive_type` characteristic for a conversation. This call overrides the value that was assigned when the `Initialize_Conversation` or `Accept_Conversation` call was issued.

This function currently has no effect and is supplied for program compatibility reasons.

RETURN VALUE

`return_code`	`CM_OK` `CM_PROGRAM_PARAMETER_CHECK`. The conversation_id is undefined or the application/API is in RESET state.

ERRORS

If an error has occurred, `return_code` will be different from `CM_OK` and the value will describe the error.

Set_Receive_Type (cmsrt)

not emulation
CPI-C

NAME

`Set_Receive_Type (cmsrt)` - set the conversations receive type

SYNOPSIS (C)

```
#include "cpic.h"

/* Set_Receive_Type */
CM_ENTRY cmsrt(
        unsigned char CM_PTR conversation_id,
        CM_INT32 CM_PTR      receive_type,
        CM_INT32 CM_PTR      return_code);
```

SYNOPSIS (PERL)

```
use Cpic;
$rc = Cpic::Set_Receive_Type($cid,$rtype);
```

DESCRIPTION

`Set_Receive_Type (cmsrt)` is used by a program to set the `receive_type` characteristic for a conversation. `Set_Receive_Type` overrides the value that was assigned when the `Initialize_Conversation` or `Accept_Conversation` call was issued

`CM_RECEIVE_AND_WAIT`	Receive will wait for data from the host, see `-RW` option.
`CM_RECEIVE_IMMEDIATE`	Receive will return immediately with the data or an error if there was none.

RETURN VALUE

`return_code`	`CM_OK` `CM_PROGRAM_PARAMETER_CHECK`. The conversation_id is undefined or the application/API is in RESET state.

ERRORS

If an error has occurred, `return_code` will be different from `CM_OK` and the value will describe the error.

Set_Send_Type (cmsst)

not emulation
CPI-C

NAME

`Set_Send_Type (cmsst)` - set the conversations send type

SYNOPSIS (C)

```
#include "cpic.h"

/* Set_Send_Type */
CM_ENTRY cmsst(
        unsigned char CM_PTR conversation_id,
        CM_INT32 CM_PTR      send_type,
        CM_INT32 CM_PTR      return_code);
```

SYNOPSIS (PERL)

```
use Cpic;
$rc = Cpic::Set_Send_Type($cid, $stype);
```

DESCRIPTION

`Set_Send_Type (cmsst)` is used by a program to set the `send_type` characteristic for a conversation. `Set_Send_Type` overrides the value that was assigned when the `Initialize_Conversation` or `Accept_Conversation` call was issued.

The default sent_type is `CM_BUFFER_DATA` which will only send the data to the host when the line buffer is full. If the `Send_Data` is to send the data directly to the host, then `CM_SEND_AND_PREP_TO_RECEIVE` should be used. See also the `-SS, -DD, -ET` options for the `cpic.cfg` file.

The possible values for the send_type are:

CM_BUFFER_DATA CM_SEND_AND_FLUSH CM_AND_CONFIRM	By default send the data with no enclosure. Can be changes with the -ET option.
CM_SEND_AND_PREP_TO_RECEIVE	Sends the data with the turn. After this call the CPI-C is in RECEIVE state.
CM_SEND_AND_DEALLOCATE	Sends the data with the turn and Deallocate the session. After this call the CPI-C is in RESET state. The -DD option.

RETURN VALUE

return_code	CM_OK CM_PROGRAM_PARAMETER_CHECK. The conversation_id is undefined or the application/API is in RESET state.

ERRORS

If an error has occurred, return_code will be different from CM_OK and the value will describe the error.

Set_Sync_Level (cmssl)

not emulation
CPI-C

NAME

`Set_Sync_Level (cmssl)` - set the sync level

SYNOPSIS (C)

```
#include "cpic.h"

/* Set_Sync_Level */
CM_ENTRY cmssl(
        unsigned char CM_PTR conversation_id,
        CM_INT32 CM_PTR      sync_level,
        CM_INT32 CM_PTR      return_code);
```

SYNOPSIS (PERL)

```
use Cpic;
$rc = Cpic::Set_Sync_Level($cid,$slevel);
```

DESCRIPTION

`Set_Sync_Level (cmssl)` is used by a program to set the `sync_level` characteristic for a given conversation. The `sync_level` characteristic is used to specify the level of synchronization processing between the two programs. It determines whether the programs support no synchronization, confirmation-level synchronization (using the Confirm and Confirmed CPI-C calls), or sync-point-level synchronization (using the calls of a resource recovery interface). `Set_Sync_Level` overrides the value that was assigned when the `Initialize_Conversation` call was issued.

This function currently has no effect and is supplied for program compatibility reasons.

RETURN VALUE

`return_code`	`CM_OK` `CM_PROGRAM_PARAMETER_CHECK`. The conversation_id is undefined or the application/API is in RESET state.

ERRORS

If an error has occurred, `return_code` will be different from `CM_OK` and the value will describe the error.

Set_TP_Name (cmstpn)

not emulation
CPI-C

NAME

`Set_TP_Name (cmstpn)` - set the remote program name

SYNOPSIS (C)

```
#include "cpic.h"

/* Set_TP_name */
CM_ENTRY cmstpn(
        unsigned char CM_PTR conversation_id,
        unsigned char CM_PTR TP_name,
        CM_INT32 CM_PTR      TP_name_len,
        CM_INT32 CM_PTR      return_code);
```

SYNOPSIS (PERL)

```
use Cpic;
$rc = Cpic::Set_TP_Name($cid, $name);
```

DESCRIPTION

`Set_TP_Name (cmstpn)` is used by a program to set the `TP_name` and `TP_name_length` characteristics for a given conversation. `Set_TP_Name` overrides the current values that were originally acquired from the side information using the `sym_dest_name`. This call does not change the values in the side information. `Set_TP_Name` only changes the `TP_name` and `TP_name_length` characteristics for this conversation.

This function currently has no effect and is supplied for program compatibility reasons.

RETURN VALUE

`return_code`	`CM_OK` `CM_PROGRAM_PARAMETER_CHECK`. The conversation_id is undefined or the application/API is in RESET state.

ERRORS

If an error has occurred, `return_code` will be different from `CM_OK` and the value will describe the error.

Set_Conversation_Security_Type (cmscst)

not emulation
CPI-C

NAME

`Set_Conversation_Security_Type (cmscst)` - set the conversation security type (same as `xcscst`)

SYNOPSIS (C)

```
#include "cpic.h"

/* Set_Conversation_Security_Type */
CM_ENTRY cmscst(
        unsigned char CM_PTR conversation_id,
        CM_INT32 CM_PTR      conv_sec_type,
        CM_INT32 CM_PTR      return_code);
```

SYNOPSIS (PERL)

```
use Cpic;
$rc = Cpic::Set_Conversation_Security_Type($cid,$t);
```

DESCRIPTION

`Set_Conversation_Security_Type (cmscst)` is used by a program to set the `conversation_security_type` characteristic for a conversation. `Set_Conversation_Security_Type` overrides the current value, which was originally acquired from the side information using `sym_dest_name`.

This function currently has no effect and is supplied for program compatibility reasons.

RETURN VALUE

`return_code`	`CM_OK` `CM_PROGRAM_PARAMETER_CHECK`. The conversation_id is undefined or the application/API is in RESET state.

ERRORS

If an error has occurred, `return_code` will be different from `CM_OK` and the value will describe the error.

Set_Conversation_Security_User_ID (cmscsu)

not emulation
CPI-C

NAME

`Set_Conversation_Security_User_ID (cmscsu)` - set the conversations security id

SYNOPSIS (C)

```
#include "cpic.h"

/* Set_Conversation_Security_User_ID */
CM_ENTRY cmscsu(
        unsigned char CM_PTR conversation_id,
        unsigned char CM_PTR user_id,
        CM_INT32 CM_PTR      user_id_len,
        CM_INT32 CM_PTR      return_code);
```

SYNOPSIS (PERL)

```
use CpicXXXX;
$rc = Cpic::Set_Conversation_Security_User_ID
($cid,$uid);
```

DESCRIPTION

`Set_Conversation_Security_User_ID (cmscsu)` is used by a program to set the `security_user_ID` and `security_user_ID_length` characteristics for a conversation. `Set_Conversation_Security_User_ID` overrides the current values, which were originally acquired from the side information using `sym_dest_name`.

This function currently has no effect and is supplied for program compatibility reasons.

RETURN VALUE

`return_code`	`CM_OK` `CM_PROGRAM_PARAMETER_CHECK`. The conversation_id is undefined or the application/API is in RESET state.

ERRORS

If an error has occurred, `return_code` will be different from `CM_OK` and the value will describe the error.

Set_Conversation_Security_Password (cmscsp)

not emulation
CPI-C

NAME

`Set_Conversation_Security_Password (cmscsp)` - set the conversations security password id

SYNOPSIS (C)

```
#include "cpic.h"

/* Set_Conversation_Security_Password */
CM_ENTRY cmscsp(
        unsigned char CM_PTR conversation_id,
        unsigned char CM_PTR password_id,
        CM_INT32 CM_PTR      password_id_len,
        CM_INT32 CM_PTR      return_code);
```

SYNOPSIS (PERL)

```
use Cpic;
$rc = Cpic::Set_Conversation_Security_Password($cid,$pw);
```

DESCRIPTION

`Set_Conversation_Security_Password (cmscsp)` is used by a program to set the `security_password` and `security_password_length` characteristics for a conversation. `Set_Conversation_Security_Password` overrides the current values, which were originally acquired from the side information using `sym_dest_name`.

This function currently has no effect and is supplied for program compatibility reasons.

RETURN VALUE

`return_code`	`CM_OK` `CM_PROGRAM_PARAMETER_CHECK`. The conversation_id is undefined or the application/API is in RESET state.

ERRORS

If an error has occurred, `return_code` will be different from `CM_OK` and the value will describe the error.

Set_CPIC_Side_Information (xcmssi)

not emulation
CPI-C

NAME

`Set_CPIC_Side_Information (xcmssi)` - set side information

SYNOPSIS (C)

```
#include "cpic.h"

/* Set_CPIC_Side_Information */
CM_ENTRY xcmssi(
        unsigned char CM_PTR key_lock,
        SIDE_INFO CM_PTR     side_info_entry,
        CM_INT32 CM_PTR      side_info_length,
        CM_INT32 CM_PTR      return_code);
```

SYNOPSIS (PERL)

```
N/A
```

DESCRIPTION

`Set_CPIC_Side_Information (xcmssi)` is used by a program to set the side information. `Set_CPIC_Side_Information` overrides the current values, which were originally acquired from the side information using `sym_dest_name`. The `sym_dest_name` will be used by the next call to `Accept_Conversation` to retrieve the line handler parameters to wait for the incoming connection.

```
#ifdef windows
WORD    wVer = 2;
WCPICDATA       CPICData;

    if (WinCPICStartup(wVer, &CPICData) != 0)
        return;
#endif

    memset(&side_info, 0, sizeof(side_info));

/* set name to look for in the cpic.cfg file */
    strncpy(side_info.sym_dest_name, "dps8in",
                        sizeof(side_info.sym_dest_name));
```

```
    len = (CM_INT32) sizeof(side_info);
    Set_CPIC_Side_Information("", &side_info, &len, &retcode);

/* wait for the incoming connection */
    Accept_Conversation(conv_id, &retcode);
```

RETURN VALUE

`return_code`	`CM_OK` `CM_PROGRAM_PARAMETER_CHECK`. The conversation_id is undefined or the application/API is in RESET state.

ERRORS

If an error has occurred, `return_code` will be different from `CM_OK` and the value will describe the error.

Set_Conversation_Security_Type (xcscst)

not emulation CPI-C

NAME

`Set_Conversation_Security_Type (xcscst)` - set the conversation security type (same as cmscst)

SYNOPSIS (C)

```
#include "cpic.h"

/* Set_Conversation_Security_Type */
CM_ENTRY xcscst(
        unsigned char CM_PTR conversation_id,
        CM_INT32 CM_PTR      conv_sec_type,
        CM_INT32 CM_PTR      return_code);
```

SYNOPSIS (PERL)

```
use CpicXXXX;
$rc = CpicXXXX::Set_Conversation_Security_Type
($cid,$stype);
```

DESCRIPTION

`Set_Conversation_Security_Type (xcscst)` is used by a program to set the `conversation_security_type` characteristic for a conversation. `Set_Conversation_Security_Type` overrides the current value, which was originally acquired from the side information using `sym_dest_name`.

This function currently has no effect and is supplied for program compatibility reasons.

RETURN VALUE

`return_code`	`CM_OK` `CM_PROGRAM_PARAMETER_CHECK`. The conversation_id is undefined or the application/API is in RESET state.

ERRORS

If an error has occurred, `return_code` will be different from `CM_OK` and the value will describe the error.

WinCPICIsBlocking

NAME

`WinCPICIsBlocking` - Windows CPI-C call to test if blocking

SYNOPSIS (C)

Windows only

```
#include "cpic.h"

BOOL WINAPI WinCPICIsBlocking(void);
```

SYNOPSIS (PERL)

```
N/A
```

DESCRIPTION

`WinCPICIsBlocking` determines whether the executing task is waiting for a previous synchronous call to finish.

This function currently has no effect and is supplied for program compatibility reasons.

RETURN VALUE

`return`	`TRUE` Task is waiting for an outstanding call.

ERRORS

If an error has occurred the `return code` will be `FALSE`.

WinCPICSetBlockingHook

NAME

`WinCPICSetBlockingHook` - Windows CPI-C call set blocking hook

SYNOPSIS (C)

Windows only

```
#include "cpic.h"

FARPROC WINAPI WinCPICSetBlockingHook(
        FARPROC lpfnBlockingHook);
```

SYNOPSIS (PERL)

```
N/A
```

DESCRIPTION

`WinCPICSetBlockingHook` allows a program to set a Hooking function which will be called during blocking calls.

This function currently has no effect and is supplied for program compatibility reasons.

RETURN VALUE

`return`	`TRUE` Successful

ERRORS

If an error has occurred the `return code` will be `FALSE`.

WinCPICUnhookBlockingHook

NAME

`WinCPICUnhookBlockingHook` - Windows CPI-C call removes blocking hook

SYNOPSIS (C)

Windows only

```
#include "cpic.h"

BOOL WINAPI WinCPICUnhookBlockingHook(void);
```

SYNOPSIS (PERL)

```
N/A
```

DESCRIPTION

`WinCPICUnhookBlockingHook` allows a program to reset a Hooking function which was set with the WinCPICSetBlockingHook call.

This function currently has no effect and is supplied for program compatibility reasons.

RETURN VALUE

`return`	`TRUE` Successful.

ERRORS

If an error has occurred the `return code` will be `FALSE`.

WinCPICStartup

NAME

`WinCPICStartup` - Windows CPI-C initialization call

SYNOPSIS (C)

Windows only

```
#include "cpic.h"

int WINAPI WinCPICStartup(
        WORD wVersionRequired,
        LPWCPICDATA lpwCPICData);
```

SYNOPSIS (PERL)

```
N/A
```

DESCRIPTION

`WinCPICStartup` allows a program to specify a version for the Window CPI-C DLL. The DLL will return details. This function must be called once as the first call to the Windows CPI-C DLL.

RETURN VALUE

`return`	`0` Successfully initialized

ERRORS

If an error has occurred the `return code` will be different to 0 and the value will describe the error.

WinCPICCleanup

NAME

`WinCPICCleanup` - Windows CPI-C cleanup termination call

SYNOPSIS (C)

Windows only

```
#include "cpic.h"

BOOL WINAPI WinCPICCleanup(void);
```

SYNOPSIS (PERL)

```
N/A
```

DESCRIPTION

`WinCPICCleanup` terminates and deallocated internal resources. This function must be called once as the last call to the Windows CPI-C DLL.

RETURN VALUE

`return`	`TRUE` Successfully terminated.

ERRORS

If an error has occurred the `return code` will be `FALSE`.

api_msg

NAME

`api_msg` - delivers a message based on the CPI-C retcode.

SYNOPSIS (C)

```
#include "cpic.h"

char * api_msg (int   keymsg, char *dummyfile)
```

SYNOPSIS (PERL)

```
use CpicXXXX;
$message = CpicXXXX::Api_msg ($keymsg);
```

DESCRIPTION

When `api_msg()` is called with a return code from another CPI-C function in the keymsg variable, it will return a text message describing the return code in question.

This function is not X/Open compliant and is only available when linking to the static libraries. It is not available in the Windows CPI-C DLLs.

RETURN VALUE

The format of the returned text message is:

`message identifier`	4 digits
`filler`	3 spaces
`message text`	80 characters
`end of string`	1 byte (null character)

CPI-C 3270: keyboard input

CPI-C 3270 API supports the function key definitions defined in the IBM HLLAPI interface. It uses '@' as an escape key followed by a mnemonic code that corresponds to the supported host function. An example is PF1, which is coded as @1.

The application should fill the Send_Data buffer with characters and functions keys in the order they would have been typed in by a user. This means that the key initiating a send (ENTER, PFx, PAx or CLEAR) should be coded at the end of the buffer delivered to Send_Data. Within the buffer you may use TAB, HOME, BACKSPACE or other local functions. Below you will find a complete list of functions keys currently supported:

Meaning	Mnemonic
@	@@
Backspace	@<
Backtab (Left Tab)	@B
Clear	@C
Cursor Down	@V
Cursor Left	@L
Cursor Right	@Z
Cursor Select	@A@J
Cursor Up	@U
Delete	@D
Dup	@S@x
Enter	@E
Erase EOF	@F
Erase Input (EOP)	@G
Erase Input (EOP)	@A@F
Field Mark	@S@y

Meaning	Mnemonic
Home	@0 (zero)
Insert Mode	@I
Left Tab (Back Tab)	@B
New Line	@N
Reset	@R
Right Tab (Tab)	@T
Sys Request **Note: Used as Break key**	@A@H
Tab (Right Tab)	@T
PA1	@x
PA2	@y
PA3	@z
PF1/F1	@1
PF2/F2	@2
PF3/F3	@3
PF4/F4	@4
PF5/F5	@5
PF6/F6	@6
PF7/F7	@7
PF8/F8	@8
PF9/F9	@9
PF10/F10	@a
PF11/F11	@b
PF12/F12	@c
PF13	@d
PF14	@e
PF15	@f

Meaning	Mnemonic
PF16	@g
PF17	@h
PF18	@i
PF19	@j
PF20	@k
PF21	@l
PF22	@m
PF23	@n
PF24	@o

NOTE: If you want to use the "at" sign (@) in the Data String, you must use the two-byte code "@@".

CPI-C 5250: keyboard input

CPI-C 5250 API supports the function key definitions defined in the IBM HLLAPI interface. It uses '@' as an escape key followed by a mnemonic code that corresponds to the supported host function. An example is PF1, which is coded as @1.

The application should fill the Send_Data buffer with characters and functions keys in the order they would have been typed in by a user. This means that the key initiating a send (ENTER, PFx, PAx or CLEAR) should be coded at the end of the buffer delivered to Send_Data. Within the buffer you may use TAB, HOME, BACKSPACE or other local functions. Below you will find a complete list of functions keys currently supported:

Meaning	Mnemonic
@	@@
Backspace	@<
Backtab (Left Tab)	@B
Clear	@C
Cursor Down	@V
Cursor Left	@L
Cursor Right	@Z
Cursor Select	@A@J
Cursor Up	@U
Delete	@D
Dup	@S@x
Enter	@E
Erase EOF	@F
Erase Input (EOP)	@G
Erase Input (EOP)	@A@F
Field Mark	@S@y

Meaning	Mnemonic
Home	@0 (zero)
Insert Mode	@I
Left Tab (Back Tab)	@B
New Line	@N
Reset	@R
Right Tab (Tab)	@T
Sys Request **Note: Used as Break key**	@A@H
Tab (Right Tab)	@T
PA1	@x
PA2	@y
PA3	@z
PF1/F1	@1
PF2/F2	@2
PF3/F3	@3
PF4/F4	@4
PF5/F5	@5
PF6/F6	@6
PF7/F7	@7
PF8/F8	@8
PF9/F9	@9
PF10/F10	@a
PF11/F11	@b
PF12/F12	@c
PF13	@d
PF14	@e
PF15	@f

Meaning	Mnemonic
PF16	@g
PF17	@h
PF18	@i
PF19	@j
PF20	@k
PF21	@l
PF22	@m
PF23	@n
PF24	@o

NOTE: If you want to use the "at" sign (@) in the Data String, you must use the two-byte code "@@".

CPI-C 7800: keyboard input

CPI-C 7800 API supports function key definitions in very much the same style as the *CPI-C 3270 API*. It uses '@' as an escape key followed by a mnemonic code that corresponds to the supported host function. An example is F1, which is coded as @1.

The application should fill the Send_Data buffer with characters and functions keys in the order they would have been typed in by a user. This means that the key initiating a send (i.e. XMIT, F1-F12) should be coded at the end of the buffer delivered to Send_Data. Within the buffer you may use TAB, HOME, BACKSPACE or other local functions. Below you will find a complete list of functions keys currently supported:

Meaning	Mnemonic
@	@@
Backspace	@<
Backtab (Left Tab)	@B
Clear	@C
Cursor Down	@V
Cursor Left	@L
Cursor Right	@Z
Cursor Up	@U
Delete Character	@D
Host Break	@A@H
Set Function Code Character: 'c'	@A@C@c
Attribute: 'c'	@S@Ac
Delete Attribute	@S@B
Delete Line	@S@D
Insert Line	@S@I
Previous Segment in 72 line mode	@S@M

Meaning	Mnemonic
Next Segment in 72 line mode	@S@N
Paste Line	@S@P
Reset Initial State	@S@R
Tab Clear	@S@c
Fold Line	@S@f
Unfold Line	@S@u
Carriage Return	@S@r
Skip Line	@S@s
Tab Set	@S@t
Duplicate Character	@S@x
Transmit Data	@E
Transmit All	@P
Erase EOF	@F
Erase EOP	@G
Home	@0 (zero)
Insert Mode	@I
Insert Mode Reset	@O
Left Tab (Back Tab)	@B
New Line	@N
Reset	@R
Right Tab (Tab)	@T
Tab (Right Tab)	@T
F1	@1
F2	@2
F3	@3
F4	@4
F5	@5

Meaning	Mnemonic
F6	@6
F7	@7
F8	@8
F9	@9
F10	@a
F11	@b
F12	@c
Shifted F1	@d
Shifted F2	@e
Shifted F3	@f
Shifted F4	@g
Shifted F5	@h
Shifted F6	@i
Shifted F7	@j
Shifted F8	@k
Shifted F9	@l
Shifted F10	@m
Shifted F11	@n
Shifted F12	@o

NOTE: If you want to use the "at" sign (`@`) in the Data String, you must use the two-byte code "`@@`".

CPI-C DKU: keyboard input

CPI-C DKU API supports function key definitions in very much the same style as the *CPI-C 3270 API*. It uses '@' as an escape key followed by a mnemonic code that corresponds to the supported host function. An example is F1, which is coded as @1.

The application should fill the Send_Data buffer with characters and functions keys in the order they would have been typed in by a user. This means that the key initiating a send (i.e. XMIT, F1-F12) should be coded at the end of the buffer delivered to Send_Data. Within the buffer you may use TAB, HOME, BACKSPACE or other local functions. Below you will find a complete list of functions keys currently supported:

Meaning	Mnemonic
@	@@
Backspace	@<
Backtab (Left Tab)	@B
Clear	@C
Cursor Down	@V
Cursor Left	@L
Cursor Right	@Z
Cursor Up	@U
Delete Character	@D
Delete Line	@S@D
Insert Line	@S@I
Paste Line	@S@P
Clear All Tab Stops	@S@a
Back Character	@S@b
Clear Tab Stop	@S@c
Fold Line	@S@f

Meaning	Mnemonic
Insert Character	@S@i
Paste Character	@S@p
Unfold Line	@S@u
Carriage Return	@S@r
Skip Line	@S@s
Set Tab Stop	@S@t
Duplicate characer	@S@x
Transmit	@E
Transmit Page	@P
Erase EOF	@F
Erase EOP	@G
Home	@0 (zero)
Insert Mode	@I
Left Tab (Back Tab)	@B
New Line	@N
Reset	@R
Right Tab (Tab)	@T
Tab (Right Tab)	@T
F1	@1
F2	@2
F3	@3
F4	@4
F5	@5
F6	@6
F7	@7
F8	@8
F9	@9

Meaning	Mnemonic
F10	@a
F11	@b
F12	@c
Host Break	@A@H
FKC-0 Send FKC Only	@A@A@0
FKC-1 Send FKC and Screen	@A@A@1
Set Function Code Character: 'c'	@A@C@c

NOTE: If you want to use the "at" sign (@) in the Data String, you must use the two-byte code "@@".

CPI-C configuration file: cpic.cfg

The configuration file, `cpic.cfg`, will be read from the default *`Host Links`* configuration directory and the full path name is:

UNIX location	`/usr/gar/config/default/cpic.cfg`
Windows location	`\gar\config\default\cpic.cfg`

The *`Host Links`* installation program will copy the example configuration file to this directory at first time installation. This must be modified to suite your environment. Only the configuration file in the default directory is read, you cannot have private user configuration directories.

`cpic.cfg` is read sequentially and can consist of several sections which enables the application to address one of several hosts and host applications. A "Default" section can be used for parameters common to all connections, and "node hostlabel" sections can be used to group parameters specific to each host application.

The `-host` can be user to precede host connection parameters and `-user` to precede CPI-C parameters.

The null-terminated string given to `Initialize_Conversation` in the character buffer `sym_dest_name` must match the "hostlabel" string in "node hostlabel".

Below you will see an example of a `cpic.cfg` that can be used to access an IBM host application either through OSI (OSF/Janus in Datanet) or TCP/IP (TN3270):

```
* Sample cpic.cfg file
Default
    -host
    -s_
    -user
    -snd off
    -dbg
```

```
node tcpibm
    -li tcp
    -am tn3270
    -tm IBM-3278-2-e
    -ll 6144
* remember to include "-ho ipaddress" to address
   * remote host

node osiibm
    -li dsa
    -hm ibm
    -tm ibm3270
    -ll 6144
* remember to include "-da appl -dn host -lm logmode"
* to address host and application. "-dn host" is a
* reference to "rsc host" in dsa.cfg

node iofen06
    -li dsa
    -hm dps7
    -da iof
    -dn en06
    -du user
    -pw passwd

node tssdps5
    -li dsa:192.150.211.4
*   -d_ on
*   -s_ on
    -hm dps8
    -da tss
    -dn dps5
    -ur userrec
    -pw passwd

   * End of cpic.cfg
```

NOTE: '*' at the beginning of a line marks it as a comment.

In order to check your connection to the host application(s), you should, if possible, log in interactivly using the real emulators *G3270*, *G5250*, *V78sim* or *Qsim*. *G3270* uses the configuration file `g3270.cfg`, *G5250* uses the configuration file `g5250.cfg`, *V78sim* uses the configuration file `v78sim.cfg` and *Qsim* uses the configuration file `qsim.cfg`. When all parameters are set correctly and a successful connection is achieved, the line handler parameters can be copied from the command line or `g3270.cfg/g5250.cfg/v78sim.cfg/qsim.cfg` to `cpic.cfg`.

CPI-C API Parameters

Parameter	Description
-CW ON/off	Wait for log-on screen/welcome banner. Turn it off if nothing is sent by the host application after the connection has been established.
-DBG on/OFF	Turns internal tracing on. The trace file will be located in the Debug directory, unless -REP is in use.
-DD on/OFF	Tell CPI-C to disconnect the line when it gets changed to the DEALLOCATE state.
-DG on/OFF	Tells CPI-C if it should change to the DEALLOCATE state when it receives group terminated data from the host.
-ET n	Indicates the enclosure type to use when sending buffered data to the host. Default is 0 (e_none). Other values are, 1 = e_segment, 2=e_message, 3=e_group. See a description of enclosures on page 20.
-LW nn	Sets the time CPI-C will wait for an incoming call with the cmaccp function. Default is wait indefinitely.
-REP filename	Filename/path for internal tracing. See -DBG option.
-RW nn	Receive wait. Time-out value in seconds for Receive call. The default is to block until data/disconnection is received from the host application.
-S_ on/OFF	Write internal event- and data trace information to stderr.
-SND ON/off	The API is in send state after issuing a Allocate call. If turned OFF, it will be in receive state after the Allocate call.

Parameter	Description
`-SS on/OFF`	When set ON, this option makes CPI-C compatible with pre-5.0 versions of *G&R/CPI-C API Starter Set*. It sets the following initial states: Deallocate type is set to `CM_DEALLOCATE_ABEND` and Send Type is set to `CM_SEND_AND_PREPARE_TO_RECEIVE`. Also the `Receive` function will only return when it has received the turn from the host.

CPI-C Emulation API Parameters

The default value of the option, which will be used when the parameter is not specified, is written in CAPITAL letters. For complete list of emulation specific parameters see the *G3270*, *G5250*, *V78sim* and *Qsim* manuals

Parameter	Description
`-BS on/OFF`	`Receive` return on clear screen on local token. (CPI-C 3270, 7800 and DKU only)
`-DBG on/OFF`	Turns internal tracing on. The trace file will be located in the Debug directory unless `-REP` is in use.
`-REP filename`	Filename/path for internal tracing. See `-DBG` option.
`-RW nn`	Receive wait. Time-out value in seconds for `Receive` call. The default is to block until data/disconnection is received from the host application.
`-SND ON/off`	The API is in send state after issuing a `Allocate` call. If turned OFF, it will be in receive state after the `Allocate` call.

Troubleshooting

If you are experiencing any kind of problem when using any of the GlAPI CPI-C programmatic interfaces to access your host application, the CPI-C trace file and/or the line handler trace file will provide useful documentation of the problem. Either for your own use, to the G&R distributor or to G&R if it turns out to be caused by an error in the program itself. See the appendix *Host Links trace* for a full discussion of how to generate G&R/Host Links trace files.

See section entitled *Gline data and session trace* on page 144 for a description of the way the standard debug sub-directories and file names are generated.

GlAPI tracefile

When activated, the GlAPI trace routine will log every Gline API function call and the parameters passed to it by the GlAPI application to the file *'glapi.dbg',* located in the users debugging directory (e.g. /usr/gar/debug/jim/glapi.dbg').

GlAPI debugging is activated by an environment variable called 'GAR_GLAPIDEBUG' e.g. for a UNIX system the following command is necessary:

```
GAR_GLAPIDEBUG=1
export GAR_GLAPIDEBUG
```

CPI-C tracefile

UNIX location	`/usr/gar/debug/`**`XXX`**`/cp`**`A-NNNN.`**`dbg`
Windows location	`C:\gar\debug/`**`XXX`**`/cp`**`A-NNNN.`**`dbg`

(XXX = user name)
(A =API reference: i=native CPI-C, 3=3270, 5=5250, 7=7800, d=DKU)
(NNNN=process id)

This trace file contains details about the API's processing of both host input and user input. To enable this tracing, add the `-DBG` option to the relevant section of the `cpic.cfg` configuration file:

```
node TP8WS002
-li YYY
..
-user
  -dbg
```

(YYY =line handler identification, i.e. DSA, DIWS or TCP)

Line handler trace file

UNIX location	`/usr/gar/debug/`**`XXX`**`/cp`**`A-NNNN`**`.gli`
Windows location	`C:\gar\debug/`**`XXX`**`/cp`**`A-NNNN`**`.gli`

(XXX = user name)

This trace file contains details about line handler operation. To enable line handler tracing, add one or both of the `-D_` and `-S_` options to the relevant section of the `cpic.cfg` configuration file before the CPI-C -user options:

```
node TP8WS002
-li YYY
  -s_ on
  -d_ on
-user
..
```

(YYY =line handler identification, i.e. DSA, DIWS or TCP)

When connecting through Ggate

UNIX location:	`/usr/gar/debug/`**`ZZZZ`**`/gga`**`NN-PPPP`**`.dbg`
Windows location:	`C:\gar\debug\`**`ZZZZ`**`\gga`**`NN-PPPP`**`.dbg`

(ZZZZ = DSA node name, e.g. EN06 or PH13)
(NN =Instance number, starting at 01)
(PPPP =IP-address of the client system, running Qsim in this case)

When a application based on GlAPI are connecting through Ggate to the host application, the line handler trace will be generated on the Ggate system, with the name and location showed in the table above. In this case the relevant section of the `cpic.cfg` configuration file would look like this:

```
node TP8WS002
-li YYY:PPPP
  -s_ on
  -d_ on
-user
..
```

(YYY =line handler identification, i.e. DSA or DIWS)
(PPPP =IP-address of the system running Ggate)

Trace file names

The default trace file names are built using the following structure:

```
<product_id><session_id>-<process_id>.<debug_type>
```

product_id	*Value*	*Description*
	cpi	CPI-C API
	cp3	CPI-C 3270
	cp5	CPI-C 5250
	cp7	CPI-C 7800
	cpd	CPI-C DKU
session_id	(nn)	Only if multi-session application, 1-63
process_id	n (n n n...)	number of integers varies by platform
debug_type	dbg	application level debug
	gli	line trace

Example:

\gar\debug\system	debug directory for user "system"	
cpi-16.dbg	CPI-C single session debug	(-dbg)
cpi-16.gli	CPI-C single session line trace	(-li dsa -s_)
cpi2-123.dbg	CPI-C session 2 application debug	(-dbg)

Sample Gline API programs

apitest.c: One host session

```
/*-------------------------------------------------------------------

        apitest

        Purpose :       Show the use of Host Links GlAPI library to
                        handle one host session.

        Author  :       Michael Sandoy, Gallagher & Robertson A/S Norway

-------------------------------------------------------------------*/

#include <stdio.h>
#include <string.h>
#include "glapi.h"

char *parameter[] = {"-LI", "DSA", "-HM", "DPS8", NULL};
char connect_msg[] = "$*$CN TSS,VD88";
char password[] = "GAR$PASSWORD";
char lwd_cmd[] = "LWD";
char bye_cmd[] = "BYE";
char dis_cmd[] = "$*$DIS";


void display_input(void)
{
    unsigned char *p;
    enclosure_t e;
    int n;

    do {
        n = line_get(&p, &e);
        switch (e) {
        case e_attmsg:
            printf("Attention message :\n");
        default:
            p[n] = '\0';
            printf("%s\n", p);
            break;
        }
    }
    while (!line_our_turn());
}


void send_msg(char *message)
{
    printf("%s\n", message);
    line_put(message, strlen(message), e_group);
```

```
}


int main(void)
{
    int i;

    printf("*******************************************************\n");
    printf("**** Host Links GlAPI example connecting to one host ***\n");
    printf("*******************************************************\n\n");

    if (!line_initialize("", 0, "")) {
        printf("**** Error initializing GlAPI ****\n");
        return(1);
    }

    for (i = 0; parameter[i] != NULL; i++)
        if (line_init_params(parameter[i]) != 0) {
            fprintf(stderr, "%s\n", line_error);
            line_release();
            return(2);
        }
    if (line_start() != 0) {
        fprintf(stderr, "%s\n", line_error);
        line_release();
        return(2);
    }
    printf("**** Connecting to host\n");
    send_msg(connect_msg);
    display_input();
    printf("**** Sending user+password\n");
    send_msg(password);
    display_input();
    printf("**** Execute ListWorkingDirectory (LWD) command\n");
    send_msg(lwd_cmd);
    display_input();
    printf("**** Execute BYE on host\n");
    send_msg(bye_cmd);
    display_input();
    line_stop();
    line_release();

    printf("*******************************************************\n");
    printf("**** Host Links GlAPI example terminating          ****\n");
    printf("*******************************************************\n\n");
    return 0;
}

/*-----------------------------------------------------------------*/
/*-----     END : apitest                                   -----*/
/*-----------------------------------------------------------------*/
```

apitest2.c: Two host sessions

```
/*-------------------------------------------------------------------

        apitest2

        Purpose :       Show the use of Host Links GlAPI library to
                        handle two host sessions.

        Author  :       Michael Sandoy, Gallagher & Robertson A/S Norway


-------------------------------------------------------------------*/

#include <stdio.h>
#include <string.h>
#include "glapi.h"

char *parameteri[] = {"-LI", "DSA", "-HM", "DPS8",
                      "-MN", "TEST", "-LN", "IS3B", NULL};
char *parametero[] = {"-LI", "DSA", "-HM", "DPS8", NULL};
char connect_msg[] = "$*$CN TSS,VD88";
char password[] = "GAR$PASSWORD";
char dis_cmd[] = "$*$DIS";

int lid, host_lid, term_lid;
boolean h_up = FALSE;
boolean t_up = FALSE;


void display_input(void)
{
    unsigned char *p;
    enclosure_t e;
    int n;

    do {
        n = line_get(&p, &e);
        switch (e) {
        case e_attmsg:
            printf("Attention message :\n");
        default:
            p[n] = '\0';
            printf("%s\n", p);
            break;
        }
    }
    while (!line_our_turn());
}


void send_msg(char *message)
{
    printf("%s\n", message);
    line_put(message, strlen(message), e_group);
}


void process_host_input(void)
{
```

```
    unsigned char *p;
    enclosure_t e;
    int n;

    do {
        n = line_get(&p, &e);
        switch (e) {
        case e_attmsg:
            if (!line_connected())
                h_up = FALSE;
            printf("Attention message :\n");
            p[n] = '\0';
            printf("%s\n", p);
            break;
        default:
            line_switch(term_lid);
            line_put(p, n, e);
            break;
        }
        line_switch(host_lid);
    } while (e <= e_message);
    return;
}


void process_term_input(void)
{
    unsigned char *p;
    enclosure_t e;
    int n;

    do {
        n = line_get(&p, &e);
        switch (e) {
        case e_attmsg:
            if (!line_connected())
                t_up = FALSE;
            printf("Attention message :\n");
            p[n] = '\0';
            printf("%s\n", p);
            break;
        default:
            line_switch(host_lid);
            line_put(p, n, e);
            break;
        }
        line_switch(term_lid);
    } while (e <= e_message);
    return;
}


int main(void)
{
    int i;

    printf("************************************************\n");
    printf("**** Host Links GlAPI example handling two lines *\n");
    printf("************************************************\n\n");

    if (!line_initialize("", 0, "")) {
        printf("**** Error initializing GlAPI ****\n");
        return(1);
    }
```

```
printf("**** Starting line module for incoming session\n");
if (line_get_lid(&term_lid) == FALSE) {
    fprintf(stderr, "%s\n", line_error);
    line_release();
    return(2);
}
line_switch(term_lid);
for (i = 0; parameteri[i] != NULL; i++)
    if (line_init_params(parameteri[i]) != 0) {
        fprintf(stderr, "%s\n", line_error);
        line_release();
        return(2);
    }
if (line_start() != 0) {
    fprintf(stderr, "%s\n", line_error);
    line_release();
    return(2);
}
t_up = TRUE;

printf("**** Starting line module for outgoing session\n");
host_lid = term_lid + 1;
if (line_get_lid(&host_lid) == FALSE) {
    fprintf(stderr, "%s\n", line_error);
    line_release();
    return(2);
}
line_switch(host_lid);
for (i = 0; parametero[i] != NULL; i++)
    if (line_init_params(parametero[i]) != 0) {
        fprintf(stderr, "%s\n", line_error);
        line_release();
        return(2);
    }
if (line_start() != 0) {
    fprintf(stderr, "%s\n", line_error);
    line_release();
    return(2);
}
printf("**** Connecting to host\n");
send_msg(connect_msg);
display_input();
send_msg(password);
display_input();
h_up = TRUE;

while (h_up && t_up) {
    lid = line_wait(30);
    if (lid == term_lid) {
        printf("**** Terminal input received \n");
        line_switch(term_lid);
        process_term_input();
    }
    else if (lid == host_lid) {
        printf("**** Host input received \n");
        line_switch(host_lid);
        process_host_input();
    }
    else if (lid == 99) {
        fprintf(stderr, "%s\n", line_error);
        line_release();
        return(2);
    }
```

```
            else
                printf("**** Timeout !\n");
        }

        if (!t_up)
            printf("**** Terminal line was disconnected\n");
        else if (!h_up)
            printf("**** Host line was disconnected\n");

        if (h_up) {
            printf("**** Disconnecting from host\n");
            line_switch(host_lid);
            send_msg(dis_cmd);
            display_input();
        }
        if (t_up) {
            printf("**** Disconnecting from terminal\n");
            line_switch(term_lid);
            send_msg(dis_cmd);
            display_input();
        }
        line_switch(term_lid);
        line_stop();
        line_switch(host_lid);
        line_stop();
        line_release();

        printf("************************************************\n");
        printf("**** Terminating Host Links GlAPI example     ****\n");
        printf("************************************************\n\n");
        return 0;
    }

    /*--------------------------------------------------------------------*/
    /*-----     END : apitest2                                       -----*/
    /*--------------------------------------------------------------------*/
```

apiserv.c: Server session

```
/*-------------------------------------------------------------------

        apiserv

        Purpose :       Shows the use of Host Links GlAPI library to
                        accept and handle one client session from apiclnt

        Author  :       Phil Pilley, Gallagher & Robertson A/S Norway

-------------------------------------------------------------------*/

#include <stdio.h>
#include <string.h>
#include "glapi.h"

/* In both examples, the connections from GlAPI are through a remote    */
/* Ggate on the 'gars' Unix system. The syntax "-li DSA:gars" signifies */
/* That GlAPI will connect to Ggate on "gars", and use the Gline DSA    */
/* module.                                                              */

/* In the below example, the -CO is the "connection name" configured at */
/* the Ggate level, in the gline config files.                          */

char *parameters1[] = {"-li", "dsa", "-ln", "phil",
                       "-mn", "apitest", "-t_", "-s_", NULL};

char *parameters2[] = {"-li", "dsa:gars", "-ln",
                       "-mn", "apitest", NULL};

char *welcome =
        "Welcome...\n\tAnything you send will be echoplex back to you\n";


int wait_connection(void)
{
    unsigned char *p;
    enclosure_t e;
    int n;
    int found = 0;
    int i = 0;

    printf("Listening...\n");
    while (!found && i++ < 30) {
        line_wait(2);      /* wait an event for max 2 seconds */
        printf(".");
        if (line_input_available()) {
            n = line_get(&p, &e);
            printf("\n%.*s", n, p);
            if (e == e_attmsg) {
                printf("\n");
                found = 1;
            }
        }
    }
    printf("\n");
    if (!found)
        printf("Listened too long! timing out...\n");
    return(found);
```

```
}


void echo_input(void)
{
    unsigned char *p;
    enclosure_t e;
    int n;
    int found = 0;

    while (!found) {
        n = line_get(&p, &e);
        printf("\n%.*s", n, p);
        if (e == e_attmsg) {
            printf("\n");
            found = 1;
        }
        if (e == e_group) {
            printf("\n...data received...");
            found = 1;
            if (line_our_turn()) {
                printf("turn received, data echoed...\n");
                line_put(p, n, e);
            }
        }
    }
}


int main(int argc, char **argv)
{
    int i;
    char inbuf[100];
    char **params;

    printf("*********************************************************\n");
    printf("**** Listening GlAPI example, echos host reception ****\n");
    printf("*********************************************************\n\n");

    if (argc <= 1) {
        printf("**** Error You must supply Gline parameter ****\n");
        return(1);
    }

    if (**(++argv) == '-') {
       params = argv;
       printf("**** using supplied parameters ****\n");
    }
    else {
       switch (**(argv)) {
       case  '1':
          params = parameters1;
          printf("**** using parameters1 ****\n");
          break;
       case  '2':
          params = parameters2;
          printf("**** using parameters2 ****\n");
          break;
       default:
          printf("**** Error, invalid parameter number ****\n");
          return(1);
          break;
       }
    }
```

```
    if (!line_initialize("", 0, "")) {
        printf("**** Error initializing GlAPI ****\n");
        return(1);
    }

    while (*params) {
        printf("%s ", *params);
        if (line_init_params(*params) != 0) {
            fprintf(stderr, "%s\n", line_error);
            line_release();
            return(2);
        }
        params++;
    }
    printf("\n");
    if ((i = line_start()) != 0) {
        fprintf(stderr, "line_start return value = %d\n", i);
        fprintf(stderr, "%s\n", line_error);
        printf("**** ERROR ready to EXIT ?\n");
        gets(inbuf);
        line_release();
        return(2);
    }
    printf("**** Listening for host ****\n");

    if (wait_connection()) {
        line_put(welcome, strlen(welcome), e_group);

        while (line_connected())
            echo_input();
    }
    printf("**** Stopping the line interface ****\n");
    line_stop();
    line_release();

    printf("*********************************************************\n");
    printf("**** Host Links GlAPI example terminating          ****\n");
    printf("*********************************************************\n\n");
    return 0;

}

/*-------------------------------------------------------------------*/
/*-----     END : apiserv                                       -----*/
/*-------------------------------------------------------------------*/
```

apiclnt.c: Client session

```
/*-------------------------------------------------------------------

        apiclnt

        Purpose :       Shows the use of Host Links GlAPI library to
                        connect and handle one client session to apiserv

        Author  :       Phil Pilley, Gallagher & Robertson A/S Norway

-------------------------------------------------------------------*/

#include <stdio.h>
#include <string.h>
#include "glapi.h"

/* In both examples, the connections from GlAPI are through a remote    */
/* Ggate on the 'gars' Unix system. The syntax "-li DSA:gars" signifies */
/* That GlAPI will connect to Ggate on "gars", and use the Gline DSA    */
/* module.                                                              */

/* In the below example, the -CO is the "connection name" configured at */
/* the Ggate level, in the gline config files.                          */

char *parameters1[] = {"-li", "dsa", "-dn", "phil",
                       "-da", "apitest", "-t_", "-s_", NULL};

char *parameters2[] = {"-li", "dsa:gars", "-dn", "is2c",
                       "-da", "apitest", NULL};

boolean wait_line_data_and_turn(void)
{
    unsigned char *p;
    enclosure_t e;
    int n;
    int found = 0;

    while (!found) {
        line_wait(1);      /* wait an event for max 60 seconds */
        printf(".");
        if (line_input_available()) {
            n = line_get(&p, &e);
            printf("%.*s", n, p);
            if ((e == e_attmsg) || (e == e_group)) {
                printf("\n");
                found = 1;
            }
        }
    }
    return(line_connected());
}


void wait_connection(void)
{
    unsigned char *p;
    enclosure_t e;
    int n;
    int found = 0;
```

```
    printf("Connecting...\n");
    while (!found) {
        line_wait(1);      /* wait an event for max 1 seconds */
        printf(".");
        if (line_input_available()) {
            n = line_get(&p, &e);
            printf("\n%.*s", n, p);
            if (e == e_attmsg) {
                printf("\n");
                found = 1;
            }
        }
    }
    printf("\n");
}


void send_msg(char *message)
{
    int len;

    len = strlen(message);
    message[len] = '\r';
    message[++len] = '\0';

    line_put(message, len, e_group);
}


int main(int argc, char **argv)
{
    int i;
    char inbuf[100];
    char **params;

    printf("*********************************************************\n");
    printf("**** Host Links GlAPI example connecting to one host ***\n");
    printf("*********************************************************\n\n");

    if (argc <= 1) {
        printf("**** Error You must supply Gline paramerer ****\n");
        return(1);
    }

    if (**(++argv) == '-') {
       params = argv;
       printf("**** using supplied parameters ****\n");
    }
    else {
       switch (**(argv)) {
       case  '1':
          params = parameters1;
          printf("**** using parameters1 ****\n");
          break;
       case  '2':
          params = parameters2;
          printf("**** using parameters2 ****\n");
          break;
       default:
          printf("**** Error, invalid parameter number ****\n");
          return(1);
       }
    }
```

```
    if (!line_initialize("", 0, "")) {
        printf("**** Error initializing GlAPI ****\n");
        return(1);
    }

    while (*params) {
        printf("%s ", *params);
        if (line_init_params(*params) != 0) {
            fprintf(stderr, "%s\n", line_error);
            line_release();
            return(2);
        }
        params++;
    }
    printf("\n");
    if ((i = line_start()) != 0) {
        fprintf(stderr, "line_start return value = %d\n", i);
        fprintf(stderr, "%s\n", line_error);
        printf("**** ERROR ready to EXIT ?\n");
        gets(inbuf);
        line_release();
        return(2);
    }
    printf("**** Connecting to host ****\n");
    line_put("$*$CN ", 6, e_group);

    wait_connection();

    printf("**** To exit, type QUIT ****\n");

    while (line_connected()) {
        if (wait_line_data_and_turn()) {
            inbuf[0] = '\0';
            if ((gets(inbuf) == (char *)NULL) ||
                        !strcmp(inbuf, "quit") ||
                        !strcmp(inbuf, "QUIT")) {
                break;  /* asked to terminate */
            }
            send_msg(inbuf);
        }
    }
    printf("**** Stopping the line interface ****\n");
    line_stop();
    line_release();
    printf("*********************************************************\n");
    printf("**** Host Links GlAPI example terminating           ****\n");
    printf("*********************************************************\n\n");
    return 0;

}

/*---------------------------------------------------------------------*/
/*-----     END : apiclnt                                        -----*/
/*---------------------------------------------------------------------*/
```

glapitst.pl: Perl example

```
use Glapi;

# Perl example that connects to the G&R Web server and displays the first
page

# Initialization
Glapi::Initialize("", 0, "") || die;
Glapi::Parameter("-li", "tcp");

# Start the line module
$lid = Glapi::Start();

# Connect to the host
Glapi::Put("\$\*\$CN www.gar.no:80", 0, $Glapi::e_group);

Glapi::Wait(15);
while (Glapi::Input_available()) {
   # Get rid of the connect message
   ($buffer,$enc) = Glapi::Get();
   }

# Were we connected?
if (!Glapi::Connected()) {
   print "Oops - not connected.\n";
   Glapi::Stop();
   exit;
   }

# Request the top page from the www.gar.no web server
Glapi::Put("GET / HTTP/1.0\r\n\r\n", 0, $Glapi::e_group);

# Display the result from the web server
$go = Glapi::Wait(15);
while ($go == $lid) {
   while (Glapi::Input_available()) {
      ($buffer,$enc) = Glapi::Get();
      print "$buffer\n" if ($enc < $Glapi::e_attmsg);
      }
   if ($enc == 6 && !Glapi::Connected()) {
      $go = -1;
      }
   else {
      $go = Glapi::Wait(15);
      }
   }

# Deallocate resources and stop the line module
Glapi::Release();
Glapi::Stop();
```

Sample CPI-C API programs

cpicline.c: Connection to TSS on GCOS8

```
/*--------------------------------------------------------------------

        cpicline

        Purpose :       Show the use of Host Links Linehandler
                        CPI-C library

        Author  :       Michael Sandoy, Gallagher & Robertson A/S Norway

        Last changes:   Phil Pilley, G&R A/S, 15 Apr 1997
                        Changes for:
                                - Windows compatibility
                                - added new CPI-C api functions
                                - used READABLE_MACROS
--------------------------------------------------------------------*/

#include <stdio.h>
#include <string.h>
#include "cpic.h"

CONVERSATION_ID conv_id;
unsigned char inpbuf[6144];
unsigned char outbuf[160];
unsigned char node[16] = "tssdps5";
CM_INT32 retcode;
CM_INT32 inplen = 6144;
CM_INT32 sndlen = 2;
CM_INT32 reclen = 0;
CM_INT32 datarec;
CM_INT32 status;
CM_INT32 rts;
CM_INT32 sndtyp = CM_SEND_AND_PREP_TO_RECEIVE;
CM_INT32 dealtyp = CM_DEALLOCATE_ABEND;
CM_INT32 initial_send_state = 1;
extern char *api_msg(int, char *);

void display_buffer(unsigned char *buf, CM_INT32 len)
{
    int itest;
/*
 *     fprintf(stderr,
 *             "G&R CPI-C testprogram received len=%d status=%d\n",
 *             reclen, status);
 *
 * Hexadecimal format if anyone wants it?
 *     itest = 0;
```

```
 *     fprintf(stderr, "Hexadecimal format\n");
 *     while (itest < len) {
 *         fprintf(stderr, "0x%02x ", (unsigned char) buf[itest]);
 *         if ((++itest % 16) == 0)
 *             fprintf(stderr, "\n");
 *     }
 *     if ((itest % 16) != 0)
 *         fprintf(stderr, "\n");
 *
 *     fprintf(stderr, "Character format\n");
 */
    itest = 0;
    while (itest < len) {
        fprintf(stderr, "%c", buf[itest]);
        itest++;
    }
    fprintf(stderr, "\n");
}

void send_disconnect(void)
{
    if (status != CM_DEALLOCATED_ABEND) {
        /* set deallocate type to be CM_DEALLOCATE_ABEND */
        /* to force disconnect */
        Set_Deallocate_Type(conv_id, &dealtyp, &retcode);
        Deallocate(conv_id, &retcode);
        fprintf(stderr,
                "Deallocate done conv_id=%.8s api_msg=%s\n",
                conv_id, api_msg(retcode, "dummy"));
    }
#ifdef windows
    WinCPICCleanup();
    fprintf(stderr, "WinCPICCleanup done\n");
#endif
}

int receive_turn(void)
{
    status = CM_NO_STATUS_RECEIVED;
    while (status == CM_NO_STATUS_RECEIVED) {
        Receive(conv_id, inpbuf, &inplen, &datarec, &reclen,
              &status, &rts, &retcode);
        fprintf(stderr, "Receive  done conv_id=%.8s api_msg=%s\n",
                conv_id, api_msg(retcode, "dummy"));
        display_buffer(inpbuf, reclen);
        if (retcode) {
            send_disconnect();
            return (-1);
        }
    }
    return (0);
}

int send_connect(void)
{
#ifdef windows
WORD    wVer = 2;
WCPICDATA       CPICData;

    if (WinCPICStartup(wVer, &CPICData) != 0)
        return (-1);
    fprintf(stderr, "CPICStartup done\n");
#endif
```

```
    Initialize_Conversation(conv_id, node, &retcode);
    fprintf(stderr,
            "Initialize_Conversation done conv_id=%.8s api_msg=%s\n",
            conv_id, api_msg(retcode, "dummy"));
    if (retcode) {
        send_disconnect();
        return (-1);
    }

    /* after cmallc, the default state is CM_SEND, by default G&R */
    /* CPI-C will flush any incoming logon data from the host until */
    /* the turn is received */
    Allocate(conv_id, &retcode);
    fprintf(stderr, "Allocate done conv_id=%.8s api_msg=%s\n",
            conv_id, api_msg(retcode, "dummy"));
    if (retcode) {
        send_disconnect();
        return (-1);
    }

    /* set send type to be CM_SEND_AND_PREP_TO_RECEIVE, */
    /* (send with group) */
    Set_Send_Type(conv_id, &sndtyp, &retcode);

    /* if -SND OFF in cpic.cfg, then we must read in the host */
    /* logon banner */
    if (!initial_send_state)
        return (receive_turn());

    return (0);
}

int send_break(void)
{
    Send_Error(conv_id, &rts, &retcode);
    fprintf(stderr,
            "Send_Error in SEND State done conv_id=%.8s api_msg=%s\n",
            conv_id, api_msg(retcode, "dummy"));
    if (retcode) {
        send_disconnect();
        return (-1);
    }

    return (receive_turn());
}

int send_wait_for_answer(unsigned char *msg)
{
    strcpy(outbuf, msg);
    sndlen = strlen(msg);
    Send_Data(conv_id, outbuf, &sndlen, &rts, &retcode);
    fprintf(stderr, "Send_Data done conv_id=%.8s api_msg=%s\n",
            conv_id, api_msg(retcode, "dummy"));
    display_buffer(outbuf, sndlen);

    return (receive_turn());
}

int send_then_break_wait_for_answer(unsigned char *msg)
{
    strcpy(outbuf, msg);
    sndlen = strlen(msg);
    Send_Data(conv_id, outbuf, &sndlen, &rts, &retcode);
    fprintf(stderr, "Send_Data done conv_id=%.8s api_msg=%s\n",
```

```
            conv_id, api_msg(retcode, "dummy"));
    display_buffer(outbuf, sndlen);

    Send_Error(conv_id, &rts, &retcode);
    fprintf(stderr,
            "Send_Error in RECV State done conv_id=%.8s api_msg=%s\n",
            conv_id, api_msg(retcode, "dummy"));

    return (receive_turn());
}

int main(void)
{
    /* Connect to TSS on GCOS8 system, parameters in "node tssdps5" */
    /* section of cpic.cfg */

    if (send_connect())
        return (-1);

    /* Send logon data as if logon screen has been filled in */

    if (send_wait_for_answer("account\tpassword"))
        return (-1);

    /* Execute List working directory command and read response */

    if (send_wait_for_answer("LWD"))
        return (-1);

    /* Execute List files in current directory command, read response */

    if (send_wait_for_answer("cata,s,a"))
        return (-1);

    /* Execute List file "mstull" command and read response */

    if (send_wait_for_answer("list myfile"))
        return (-1);

    /* Send BREAK and read response */

    if (send_break())
        return (-1);

    /* Execute List file "mstull" command, send BREAK, read response */

    if (send_then_break_wait_for_answer("list myfile"))
        return (-1);

    /* Execute Log out command and read response */

    if (send_wait_for_answer("bye"))
        return (0);

    /* normal return code here as we'll get disconnected with "bye" */

    /* probably won't get here as I was probably disconnected in the */
    /* receive_turn() and in RESET state but if not, then disconnect */

    send_disconnect();

    return (0);
}
```

cpicserv.c: Server session

```
/*--------------------------------------------------------------------

        cpicserv

        Purpose :       Show the use of Host Links Linehandler
                        CPI-C library incoming connection from a
                        CPI-C client (cpicclnt.c).

        Author  :       Phil Pilley, G&R A/S, 15 Apr 1997

--------------------------------------------------------------------*/

#include <stdio.h>
#include <string.h>
#include "cpic.h"

CONVERSATION_ID conv_id;
unsigned char inpbuf[6144];
unsigned char outbuf[160];
unsigned char echobuf[6144];
CM_INT32 retcode;
CM_INT32 inplen = 6144;
CM_INT32 sndlen = 2;
CM_INT32 reclen = 0;
CM_INT32 datarec;
CM_INT32 status;
CM_INT32 rts;
CM_INT32 rcv_state_after_allc = 0;
CM_INT32 sndtyp;
CM_INT32 dealtyp = CM_DEALLOCATE_ABEND;
extern char *api_msg(int, char *);

/*********************************************************/
/* This example requires the following in the cpic.cfg  */
/*********************************************************/
/*      Default                                          */
/*        -li dsa                                        */
/*      * or                                             */
/*      * -li dsa:ggate_ipaddr                           */
/*                                                       */
/*      node clnt                                        */
/*        -ln node                                       */
/*        -mn cpicserv                                   */
/*                                                       */
/*      node serv                                        */
/*        -dn node                                       */
/*        -da cpicserv                                   */
/*********************************************************/
/* NOTE that the cpicserv.c sets the "-mn" "cpicserv" in*/
/* TP_name, and will get the -ln in the cpic_cfg_name.  */
/*********************************************************/

void setup_side_info(char *cpic_cfg_name)
{
SIDE_INFO side_info;
CM_INT32 len;

    memset(&side_info, 0, sizeof(side_info));
```

```
/* set name to look for in the cpic.cfg file */
    strncpy(side_info.sym_dest_name, cpic_cfg_name,
                        sizeof(side_info.sym_dest_name));

    len = (CM_INT32) sizeof(side_info);
    Set_CPIC_Side_Information("", &side_info, &len, &retcode);
}

void display_input(char *inpbuf, CM_INT32 reclen, CM_INT32 status)
{
    int itest;

    fprintf(stderr,
            "\nG&R CPI-C testprogram receivedlen=%d status=%d\n",
            reclen, status);
/* Hexadecimal format if anyone wants it?
 *    itest = 0;
 *    fprintf(stderr, "\nHexadecimal format\n");
 *    while (itest < reclen) {
 *        fprintf(stderr, "0x%02x ", (unsigned char) inpbuf[itest]);
 *        if ((++itest % 16) == 0)
 *            fprintf(stderr, "\n");
 *    }
 *    if ((itest % 16) != 0)
 *        fprintf(stderr, "\n");
 *
 *    fprintf(stderr, "\nCharacter format\n");
 */
    itest = 0;
    while (itest < reclen) {
        fprintf(stderr, "%c", inpbuf[itest]);
        itest++;
    }
}

int send_disconnect(void)
{
    if (retcode == CM_DEALLOCATED_ABEND)
        fprintf(stderr, "\nCPIC conversation deallocated by peer\n");
    else {
        Set_Deallocate_Type(conv_id, &dealtyp, &retcode);
        Deallocate(conv_id, &retcode);
        fprintf(stderr, "\nCMDEAL done conv_id=%.8s api_msg=%s\n",
                conv_id, api_msg(retcode, "dummy"));
    }

#ifdef windows
    WinCPICCleanup();
    fprintf(stderr, "WinCPICCleanup done\n");
#endif

    return(0);
}

int wait_for_answer(char *msg)
{
int found = 0;

    if (!msg || *msg == '\0')
        return(0);

    status = CM_NO_STATUS_RECEIVED;     /* must wait for the turn */
    while (!found && status == CM_NO_STATUS_RECEIVED) {
```

```
        fprintf(stderr, "\nwaiting for: %s", msg);
        Receive(conv_id, inpbuf, &inplen, &datarec, &reclen,
                                          &status, &rts, &retcode);
        fprintf(stderr, "\nCMRCV  done conv_id=%.8s api_msg=%s",
                conv_id, api_msg(retcode, "dummy"));
        display_input(inpbuf, reclen, status);
        if (retcode != CM_OK) {
            send_disconnect();
            return (-1);
        }
        if (reclen != 0) {
            inpbuf[reclen] = '\0';
            if (!msg || !*msg || strstr(inpbuf, msg))
                found++;
        }
    }
    return (0);
}

int listen_wait(char *cpicnode, char *waitfor)
{
#ifdef windows
WORD    wVer = 2;
WCPICDATA       CPICData;

    if (WinCPICStartup(wVer, &CPICData) != 0)
        return (-1);
    fprintf(stderr, "\nCPICStartup done\n");
#endif

    setup_side_info(cpicnode);

    Accept_Conversation(conv_id, &retcode);
    fprintf(stderr, "\nCMACCP done conv_id=%.8s api_msg=%s",
            conv_id, api_msg(retcode, "dummy"));
    if (retcode) {
        return (-1);
    }
    return(wait_for_answer(waitfor));
}

int send_break(void)
{
    Send_Error(conv_id, &rts, &retcode);
    fprintf(stderr,
            "\nCMSERR in SEND State done conv_id=%.8s api_msg=%s",
            conv_id, api_msg(retcode, "dummy"));
    if (retcode) {
        send_disconnect();
        return (-1);
    }
    return(wait_for_answer(""));
}

int send_wait_for_answer(char *msg, char *waitfor)
{
    sndtyp = CM_SEND_AND_PREP_TO_RECEIVE;
    Set_Send_Type(conv_id, &sndtyp, &retcode);

    strcpy(outbuf, msg);
    sndlen = strlen(msg);
    fprintf(stderr, "\nsending: %s", msg);
    Send_Data(conv_id, outbuf, &sndlen, &rts, &retcode);
```

```
        fprintf(stderr, "\nCMSEND done conv_id=%.8s api_msg=%s",
                conv_id, api_msg(retcode, "dummy"));
        if (retcode) {
            send_disconnect();
            return (-1);
        }
        return(wait_for_answer(waitfor));
    }

    int multi_send_wait_for_answer(char *waitfor)
    {
    char org_msg[21] = "01234567890123456789";
    char msg[11];
    int  i = 0;

        while (i < 10) {
            sndtyp = (i == 9) ? CM_SEND_AND_PREP_TO_RECEIVE :
                                            CM_BUFFER_DATA;
            Set_Send_Type(conv_id, &sndtyp, &retcode);

            memcpy(msg, &org_msg[i], 10);
            msg[10] = '\0';
            strcpy(outbuf, msg);
            sndlen = strlen(msg);
            fprintf(stderr, "\nsending: %s", msg);
            Send_Data(conv_id, outbuf, &sndlen, &rts, &retcode);

            fprintf(stderr, "\nCMSEND done conv_id=%.8s api_msg=%s",
                    conv_id, api_msg(retcode, "dummy"));
            if (retcode) {
                send_disconnect();
                return (-1);
            }
            i++;
        }
        return(wait_for_answer(waitfor));
    }

    int send_then_break_wait_for_answer(char *msg, char *waitfor)
    {
        sndtyp = CM_SEND_AND_PREP_TO_RECEIVE;
        Set_Send_Type(conv_id, &sndtyp, &retcode);

        strcpy(outbuf, msg);
        sndlen = strlen(msg);
        fprintf(stderr, "\nsending: %s", msg);
        Send_Data(conv_id, outbuf, &sndlen, &rts, &retcode);

        fprintf(stderr, "\nCMSEND done conv_id=%.8s api_msg=%s",
                conv_id, api_msg(retcode, "dummy"));
        Send_Error(conv_id, &rts, &retcode);
        fprintf(stderr,
                "\nCMSERR in RECV State done conv_id=%.8s api_msg=%s",
                conv_id, api_msg(retcode, "dummy"));
        if (retcode) {
            send_disconnect();
            return (-1);
        }
        return(wait_for_answer(waitfor));
    }

    void cpicserv(char *cpicnode)
    {
```

```
    if (listen_wait(cpicnode, "CPICCLNT calling") == -1)
        return;

    if (send_wait_for_answer("CPICSERV replying", "hello") == -1)
        return;

    if (send_wait_for_answer("thanks", "012345") == -1)
        return;

    if (multi_send_wait_for_answer("finished") == -1)
        return;

    if (send_wait_for_answer("I've also finished", "nothing") == -1)
        return;

    send_disconnect();

    return;
}

int main(int argc, char **argv)
{
char *cpicnode;

    if (argc >= 2)
        cpicnode = *(++argv);
    else
        cpicnode = "clnt";

    cpicserv(cpicnode);

    return 0;
}
```

cpicclnt.c: Client session

```
/*--------------------------------------------------------------------

        cpicclnt

        Purpose :       Show the use of Host Links Linehandler
                        CPI-C library outgoing connection to a
                        CPI-C server (cpicserv.c)

        Author  :       Phil Pilley, G&R A/S, 15 Apr 1997

--------------------------------------------------------------------*/

#include <stdio.h>
#include <string.h>
#include "cpic.h"

CONVERSATION_ID conv_id;
unsigned char inpbuf[6144];
unsigned char outbuf[160];
CM_INT32 retcode;
CM_INT32 inplen = 6144;
CM_INT32 sndlen = 2;
CM_INT32 reclen = 0;
CM_INT32 datarec;
CM_INT32 status;
CM_INT32 rts;
CM_INT32 rcv_state_after_allc = 0;
CM_INT32 sndtyp;
CM_INT32 dealtyp = CM_DEALLOCATE_ABEND;
CM_INT32 synclevel = CM_NONE;
unsigned char userid[10] = "user";
CM_INT32 userlen = 4;
unsigned char password[10] = "pass";
CM_INT32 passlen = 4;
extern char *api_msg(int, char *);

/*******************************************************/
/* This example requires the following in the cpic.cfg  */
/*******************************************************/
/*      Default                                        */
/*        -li dsa                                      */
/*      * or                                           */
/*      * -li dsa:ggate_ipaddr                         */
/*                                                     */
/*      node clnt                                      */
/*        -ln node                                     */
/*        -mn cpicserv                                 */
/*                                                     */
/*      node serv                                      */
/*        -dn node                                     */
/*        -da cpicserv                                 */
/*******************************************************/
/* NOTE that the cpicclnt.c sets the "-da" "cpicserv" in*/
/* TP_name, and will get the -dn in the cpic_cfg_name.  */
/*******************************************************/
```

```
void display_input(char *inpbuf, CM_INT32 reclen, CM_INT32 status)
{
    int itest;

    fprintf(stderr,
            "\nG&R CPI-C testprogram receivedlen=%d status=%d\n",
            reclen, status);
/* Hexadecimal format if anyone wants it?
 *    itest = 0;
 *    fprintf(stderr, "\nHexadecimal format\n");
 *    while (itest < reclen) {
 *        fprintf(stderr, "0x%02x ", (unsigned char) inpbuf[itest]);
 *        if ((++itest % 16) == 0)
 *            fprintf(stderr, "\n");
 *    }
 *    if ((itest % 16) != 0)
 *        fprintf(stderr, "\n");
 *
 *    fprintf(stderr, "\nCharacter format\n");
 */
    itest = 0;
    while (itest < reclen) {
        fprintf(stderr, "%c", inpbuf[itest]);
        itest++;
    }
}

int send_disconnect(void)
{
    Set_Deallocate_Type(conv_id, &dealtyp, &retcode);
    Deallocate(conv_id, &retcode);
    fprintf(stderr, "\nCMDEAL done conv_id=%.8s api_msg=%s\n",
            conv_id, api_msg(retcode, "dummy"));

#ifdef windows
    WinCPICCleanup();
    fprintf(stderr, "WinCPICCleanup done\n");
#endif

    return(0);
}

int wait_for_answer(char *msg)
{
int found = 0;

    if (!msg || *msg == '\0')
        return(0);

    status = CM_NO_STATUS_RECEIVED;     /* must wait for the turn */
    while (!found && status == CM_NO_STATUS_RECEIVED) {
        fprintf(stderr, "\nwaiting for: %s", msg);
        Receive(conv_id, inpbuf, &inplen, &datarec, &reclen,
                                          &status, &rts, &retcode);
        fprintf(stderr,
                "\nCMRCV  done reclen=%d conv_id=%.8s api_msg=%s",
                reclen, conv_id, api_msg(retcode, "dummy"));
        display_input(inpbuf, reclen, status);
        if (retcode != CM_OK) {
            send_disconnect();
            return (-1);
        }
        if (reclen != 0) {
            inpbuf[reclen] = '\0';
```

```
            if (!msg || !*msg || strstr(inpbuf, msg))
                found++;
        }
    }
    return (0);
}

int send_connect(char *name, char *msg, char *waitfor)
{
#ifdef windows
WORD    wVer = 2;
WCPICDATA       CPICData;

    if (WinCPICStartup(wVer, &CPICData) != 0)
        return (-1);
    fprintf(stderr, "\nCPICStartup done\n");
#endif

    Initialize_Conversation(conv_id, name, &retcode);
    fprintf(stderr, "\nCMINIT done conv_id=%.8s api_msg=%s",
            conv_id, api_msg(retcode, "dummy"));
    if (retcode) {
        return (-1);
    }

    Allocate(conv_id, &retcode);
    fprintf(stderr, "\nCMALLC done conv_id=%.8s api_msg=%s",
            conv_id, api_msg(retcode, "dummy"));
    if (retcode) {
        send_disconnect();
        return (-1);
    }
    sndtyp = CM_SEND_AND_PREP_TO_RECEIVE;
    Set_Send_Type(conv_id, &sndtyp, &retcode);

    strcpy(outbuf, msg);
    sndlen = strlen(msg);
    fprintf(stderr, "\nsending: %s", msg);
    Send_Data(conv_id, outbuf, &sndlen, &rts, &retcode);

    fprintf(stderr, "\nCMSEND done conv_id=%.8s api_msg=%s",
            conv_id, api_msg(retcode, "dummy"));
    if (retcode) {
        send_disconnect();
        return (-1);
    }
    return(wait_for_answer(waitfor));
}

int send_break(void)
{
    Send_Error(conv_id, &rts, &retcode);
    fprintf(stderr,
            "\nCMSERR in SEND State done conv_id=%.8s api_msg=%s",
            conv_id, api_msg(retcode, "dummy"));
    if (retcode) {
        send_disconnect();
        return (-1);
    }
    return(wait_for_answer(""));
}

int send_wait_for_answer(char *msg, char *waitfor)
{
```

```
    sndtyp = CM_SEND_AND_PREP_TO_RECEIVE;
    Set_Send_Type(conv_id, &sndtyp, &retcode);

    strcpy(outbuf, msg);
    sndlen = strlen(msg);
    fprintf(stderr, "\nsending: %s", msg);
    Send_Data(conv_id, outbuf, &sndlen, &rts, &retcode);

    fprintf(stderr, "\nCMSEND done conv_id=%.8s api_msg=%s",
            conv_id, api_msg(retcode, "dummy"));
    if (retcode) {
        send_disconnect();
        return (-1);
    }
    return(wait_for_answer(waitfor));
}

int multi_send_wait_for_answer(char *waitfor)
{
char org_msg[21] = "01234567890123456789";
char msg[11];
int  i = 0;

    while (i < 10) {
        sndtyp = (i == 9) ? CM_SEND_AND_PREP_TO_RECEIVE :
                                        CM_BUFFER_DATA;
        Set_Send_Type(conv_id, &sndtyp, &retcode);

        memcpy(msg, &org_msg[i], 10);
        msg[10] = '\0';
        strcpy(outbuf, msg);
        sndlen = strlen(msg);
        fprintf(stderr, "\nsending: %s", msg);
        Send_Data(conv_id, outbuf, &sndlen, &rts, &retcode);

        fprintf(stderr, "\nCMSEND done conv_id=%.8s api_msg=%s",
                conv_id, api_msg(retcode, "dummy"));
        if (retcode) {
            send_disconnect();
            return (-1);
        }
        i++;
    }
    return(wait_for_answer(waitfor));
}

int send_then_break_wait_for_answer(char *msg, char *waitfor)
{
    sndtyp = CM_SEND_AND_PREP_TO_RECEIVE;
    Set_Send_Type(conv_id, &sndtyp, &retcode);

    strcpy(outbuf, msg);
    sndlen = strlen(msg);
    fprintf(stderr, "\nsending: %s", msg);
    Send_Data(conv_id, outbuf, &sndlen, &rts, &retcode);

    fprintf(stderr, "\nCMSEND done conv_id=%.8s api_msg=%s",
            conv_id, api_msg(retcode, "dummy"));
    Send_Error(conv_id, &rts, &retcode);
    fprintf(stderr,
            "\nCMSERR in RECV State done conv_id=%.8s api_msg=%s",
            conv_id, api_msg(retcode, "dummy"));
    if (retcode) {
        send_disconnect();
```

```
            return (-1);
        }
        return(wait_for_answer(waitfor));
    }

    void cpicclnt(char *cpicnode)
    {
        if (send_connect(cpicnode, "CPICCLNT calling\r",
                                    "CPICSERV replying") == -1)
            return;

        if (send_wait_for_answer("hello", "thanks") == -1)
            return;

        if (multi_send_wait_for_answer("012345") == -1)
            return;

        if (send_wait_for_answer("I'm finished", "finished") == -1)
            return;

        send_disconnect();

        return;
    }

    int main(int argc, char **argv)
    {
    char *cpicnode;

        if (argc >= 2)
            cpicnode = *(++argv);
        else
            cpicnode = "serv";

        cpicclnt(cpicnode);

        return 0;
    }
```

Sample CPI-C 3270 API programs

cpictest.c: Connection to IBM host

```
/*--------------------------------------------------------------------

        cpictest

        Purpose :       Show the use of Host Links CPI-C 3270 library

        Author  :       Michael Sandoy, Gallagher & Robertson A/S Norway


--------------------------------------------------------------------*/

#include <stdio.h>
#include <string.h>
#include "cpic.h"

CONVERSATION_ID conv_id;
char inpbuf[1920];
char outbuf[160];
char node[16] = "tcpibm";
int retcode;
int inplen = 1920;
int sndlen = 2;
int reclen = 0;
int datarec;
int status;
int rts;
int rcv_state_after_allc = 0;


void display_input(hexdisplay, inpbuf, reclen, status)
int hexdisplay;
char *inpbuf;
int reclen;
int status;
{
    int itest;

    itest = 0;
    fprintf(stderr,
            "\nCPI-C testprogram received len=%d status=%d\n",
            reclen, status);
    if (hexdisplay) {
        while (itest < reclen) {
            fprintf(stderr, "0x%02x ", inpbuf[itest]);
            if ((++itest % 16) == 0)
```

```
                fprintf(stderr, "\n");
            }
            if ((itest % 16) != 0)
                fprintf(stderr, "\n");
        }
        else {
            while (itest < reclen) {
                fprintf(stderr, "%c", inpbuf[itest]);
                if ((++itest % 80) == 0)
                    fprintf(stderr, "\n");
            }
            if ((itest % 80) != 0)
                fprintf(stderr, "\n");
        }

    }

    void disconnect()
    {
        cmdeal(conv_id, &retcode);
        fprintf(stderr,
                "\ncmdeal done conv_id=%s retcode=%d\n",
                conv_id, retcode);
    }

    int connect()
    {

        cminit(conv_id, node, &retcode);
        fprintf(stderr, "\ncminit done conv_id=%s retcode=%d",
                conv_id, retcode);
        if (retcode) {
            disconnect();
            return (-1);
        }

        cmallc(conv_id, &retcode);
        fprintf(stderr,
                "\ncmallc done conv_id=%s retcode=%d",
                conv_id, retcode);
        if (retcode) {
            disconnect();
            return (-1);
        }

        if (rcv_state_after_allc) {
            cmrcv(conv_id, inpbuf, &inplen, &datarec, &reclen,
                                            &status, &rts, &retcode);
            fprintf(stderr, "\ncmrcv  done conv_id=%s retcode=%d",
                    conv_id, retcode);
            display_input(0, inpbuf, reclen, status);
            if (retcode) {
                disconnect();
                return (-1);
            }
        }
        return (0);
    }

    int send_wait_for_answer(msg)
    char *msg;
    {
        strcpy(outbuf, msg);
        sndlen = strlen(msg);
```

```
    cmsend(conv_id, outbuf, &sndlen, &rts, &retcode);
    fprintf(stderr, "\ncmsend done conv_id=%s retcode=%d",
            conv_id, retcode);
    cmrcv(conv_id, inpbuf, &inplen, &datarec, &reclen,
                                &status, &rts, &retcode);
    fprintf(stderr, "\ncmrcv  done conv_id=%s retcode=%d",
            conv_id, retcode);
    display_input(0, inpbuf, reclen, status);
    if (retcode) {
        disconnect();
        return (-1);
    }
    return (0);
}

int main()
{
    if (connect())
        return 1;
    if (send_wait_for_answer("1@E"))
        return 1;
    if (send_wait_for_answer("userid@Tpassword@E"))
        return 1;
    if (send_wait_for_answer("@b"))     /*PF11 */
        return 1;
    if (send_wait_for_answer("@c"))     /*PF12 */
        return 1;
    disconnect();
    return 0;
}
```

cpictst.pl: Perl example

```
use Cpic3270;

# This example requires the following lines in the cpic.cfg configuration
file:
#   host locis
#    -li tcp
#     -ho locis.loc.gov
#     -am tn3270
#    -user
#     -snd off
#     -rw  20

# Initializing connection with LOCIS
$host = "locis";
($cid, $rc) = cpic3270::Initialize_Conversation ($host);
die if ($rc);

print "Connecting to $host...\n";
$rc = cpic3270::Allocate ($cid);
die if ($rc);

# Receive the first screen from Locis
($buffer, $dr, $status, $rts, $rc) = cpic3270::Receive ($cid);
# print "Receive status: $rc, dr=$dr, status=$status, rts=$rts, rcvlen=",
length($buffer),", buffer:\n, $buffer\n";

print "Requesting copyright information...\n";
# Ask for option 1 on the screen (Copyright information)
$rc = cpic3270::Send ($cid, "1\@E", 0, $rts);
die if ($rc);

# Receive the copyright info...
($buffer, $dr, $status, $rts, $rc) = cpic3270::Receive ($cid);
print "$buffer";

cpic3270::Deallocate ($cid);
```

Sample CPI-C 5250 API programs

cpictest.c: Connection to IBM host

```
/*--------------------------------------------------------------------

        cpictest

        Purpose :       Show the use of Host Links CPI-C 3270 library

        Author  :       Michael Sandoy, Gallagher & Robertson A/S Norway


--------------------------------------------------------------------*/

#include <stdio.h>
#include <string.h>
#include "cpic.h"

CONVERSATION_ID conv_id;
char inpbuf[1920];
char outbuf[160];
char node[16] = "tcpibm";
int retcode;
int inplen = 1920;
int sndlen = 2;
int reclen = 0;
int datarec;
int status;
int rts;
int rcv_state_after_allc = 0;


void display_input(hexdisplay, inpbuf, reclen, status)
int hexdisplay;
char *inpbuf;
int reclen;
int status;
{
    int itest;

    itest = 0;
    fprintf(stderr,
            "\nCPI-C testprogram received len=%d status=%d\n",
            reclen, status);
    if (hexdisplay) {
        while (itest < reclen) {
            fprintf(stderr, "0x%02x ", inpbuf[itest]);
            if ((++itest % 16) == 0)
```

```
                fprintf(stderr, "\n");
            }
            if ((itest % 16) != 0)
                fprintf(stderr, "\n");
        }
        else {
            while (itest < reclen) {
                fprintf(stderr, "%c", inpbuf[itest]);
                if ((++itest % 80) == 0)
                    fprintf(stderr, "\n");
            }
            if ((itest % 80) != 0)
                fprintf(stderr, "\n");
        }

    }

    void disconnect()
    {
        cmdeal(conv_id, &retcode);
        fprintf(stderr,
                "\ncmdeal done conv_id=%s retcode=%d\n",
                conv_id, retcode);
    }

    int connect()
    {

        cminit(conv_id, node, &retcode);
        fprintf(stderr, "\ncminit done conv_id=%s retcode=%d",
                conv_id, retcode);
        if (retcode) {
            disconnect();
            return (-1);
        }

        cmallc(conv_id, &retcode);
        fprintf(stderr,
                "\ncmallc done conv_id=%s retcode=%d",
                conv_id, retcode);
        if (retcode) {
            disconnect();
            return (-1);
        }

        if (rcv_state_after_allc) {
            cmrcv(conv_id, inpbuf, &inplen, &datarec, &reclen,
                                            &status, &rts, &retcode);
            fprintf(stderr, "\ncmrcv  done conv_id=%s retcode=%d",
                    conv_id, retcode);
            display_input(0, inpbuf, reclen, status);
            if (retcode) {
                disconnect();
                return (-1);
            }
        }
        return (0);
    }

    int send_wait_for_answer(msg)
    char *msg;
    {
        strcpy(outbuf, msg);
        sndlen = strlen(msg);
```

```
    cmsend(conv_id, outbuf, &sndlen, &rts, &retcode);
    fprintf(stderr, "\ncmsend done conv_id=%s retcode=%d",
            conv_id, retcode);
    cmrcv(conv_id, inpbuf, &inplen, &datarec, &reclen,
                                &status, &rts, &retcode);
    fprintf(stderr, "\ncmrcv  done conv_id=%s retcode=%d",
            conv_id, retcode);
    display_input(0, inpbuf, reclen, status);
    if (retcode) {
        disconnect();
        return (-1);
    }
    return (0);
}

int main()
{
    if (connect())
        return 1;
    if (send_wait_for_answer("1@E"))
        return 1;
    if (send_wait_for_answer("userid@Tpassword@E"))
        return 1;
    if (send_wait_for_answer("@b"))     /*PF11 */
        return 1;
    if (send_wait_for_answer("@c"))     /*PF12 */
        return 1;
    disconnect();
    return 0;
}
```

cpictst.pl: Perl example

```
use Cpic3270;

# This example requires the following lines in the cpic.cfg configuration
file:
#   host locis
#    -li tcp
#     -ho locis.loc.gov
#     -am tn3270
#    -user
#     -snd off
#     -rw  20

# Initializing connection with LOCIS
$host = "locis";
($cid, $rc) = cpic3270::Initialize_Conversation ($host);
die if ($rc);

print "Connecting to $host...\n";
$rc = cpic3270::Allocate ($cid);
die if ($rc);

# Receive the first screen from Locis
($buffer, $dr, $status, $rts, $rc) = cpic3270::Receive ($cid);
# print "Receive status: $rc, dr=$dr, status=$status, rts=$rts, rcvlen=",
length($buffer),", buffer:\n, $buffer\n";

print "Requesting copyright information...\n";
# Ask for option 1 on the screen (Copyright information)
$rc = cpic3270::Send ($cid, "1\@E", 0, $rts);
die if ($rc);

# Receive the copyright info...
($buffer, $dr, $status, $rts, $rc) = cpic3270::Receive ($cid);
print "$buffer";

cpic3270::Deallocate ($cid);
```

Sample CPI-C DKU API programs

dkuiof.c: Connection to IOF on GCOS7 host

```
/*--------------------------------------------------------------------

   dkucpic

   Purpose :       Show the use of Host Links CPI-C DKU library

   Author  :       Michael Sandoy, Gallagher & Robertson A/S Norway


--------------------------------------------------------------------*/

#include <stdio.h>
#include <string.h>
#include "cpic.h"

CONVERSATION_ID conv_id;
char inpbuf[1920];
char outbuf[160];
char node[16] = "iofen06";
int retcode;
int inplen = 1920;
int sndlen = 2;
int reclen = 0;
int datarec;
int status;
int rts;
int rcv_state_after_allc = 0;


void display_input(hexdisplay, inpbuf, reclen, status)
int hexdisplay;
char *inpbuf;
int reclen;
int status;
{
    int itest;

    itest = 0;
    fprintf(stderr, "\nCPI-C testprogram received len=%d status=%d\n",
       reclen, status);
    if (hexdisplay) {
    while (itest < reclen) {
```

```
            fprintf(stderr, "0x%02x ", inpbuf[itest]);
            if ((++itest % 16) == 0)
               fprintf(stderr, "\n");
        }
        if ((itest % 16) != 0)
            fprintf(stderr, "\n");
         }
         else {
        while (itest < reclen) {
            fprintf(stderr, "%c", inpbuf[itest]);
            if ((++itest % 80) == 0)
               fprintf(stderr, "\n");
        }
        if ((itest % 80) != 0)
            fprintf(stderr, "\n");
         }

    }

    void disconnect()
    {
        cmdeal(conv_id, &retcode);
        fprintf(stderr, "\ncmdeal done conv_id=%s retcode=%d\n", conv_id,
    retcode);
    }

    int connect()
    {
        cminit(conv_id, node, &retcode);
        fprintf(stderr, "\ncminit done conv_id=%s retcode=%d", conv_id, retcode);
        if (retcode) {
        disconnect();
        return (-1);
         }

        cmallc(conv_id, &retcode);
        fprintf(stderr, "\ncmallc done conv_id=%s retcode=%d", conv_id, retcode);
        if (retcode) {
        disconnect();
        return (-1);
         }

        if (rcv_state_after_allc) {
        cmrcv(conv_id, inpbuf, &inplen, &datarec, &reclen, &status, &rts,
              &retcode);
        fprintf(stderr, "\ncmrcv  done conv_id=%s retcode=%d", conv_id,
           retcode);
        display_input(0, inpbuf, reclen, status);
        if (retcode) {
            disconnect();
            return (-1);
        }
         }
        return (0);
    }

    int send_wait_for_answer(msg)
    char *msg;
    {
        strcpy(outbuf, msg);
        sndlen = strlen(msg);
        cmsend(conv_id, outbuf, &sndlen, &rts, &retcode);
        fprintf(stderr, "\ncmsend done conv_id=%s retcode=%d", conv_id, retcode);
```

```
    cmrcv(conv_id, inpbuf, &inplen, &datarec, &reclen, &status, &rts,
&retcode);
    fprintf(stderr, "\ncmrcv  done conv_id=%s retcode=%d", conv_id, retcode);
    display_input(0, inpbuf, reclen, status);
    if (retcode) {
   disconnect();
   return (-1);
    }
    return (0);
}

int main()
{
    if (connect())
   return 1;
    if (send_wait_for_answer("@E"))
   return 1;
    /*
    if (send_wait_for_answer("mp novice=1;@E"))
   return 1;
    */
    if (send_wait_for_answer("@E"))
   return 1;
    if (send_wait_for_answer("@E"))
   return 1;
    if (send_wait_for_answer("@Tmail;@E"))
   return 1;
    if (send_wait_for_answer("@E"))
   return 1;
    if (send_wait_for_answer("@Tbye;@E"))
   return 1;
    disconnect();
    return 0;
}
```

dkutss.c: Connection to TSS on GCOS8 host

```
/*--------------------------------------------------------------------

    dkucpic

    Purpose :       Show the use of Host Links CPI-C DKU library

    Author  :       Michael Sandoy, Gallagher & Robertson A/S Norway

--------------------------------------------------------------------*/

#include <stdio.h>
#include <string.h>
#include "cpic.h"

CONVERSATION_ID conv_id;
char inpbuf[1920];
char outbuf[160];
char node[16] = "tssdps5";
int retcode;
int inplen = 1920;
int sndlen = 2;
int reclen = 0;
int datarec;
int status;
int rts;
int rcv_state_after_allc = 0;


void display_input(hexdisplay, inpbuf, reclen, status)
int hexdisplay;
char *inpbuf;
int reclen;
int status;
{
    int itest;

    itest = 0;
    fprintf(stderr, "\nCPI-C testprogram receivedl en=%d status=%d\n",
        reclen, status);
    if (hexdisplay) {
    while (itest < reclen) {
        fprintf(stderr, "0x%02x ", inpbuf[itest]);
        if ((++itest % 16) == 0)
       fprintf(stderr, "\n");
    }
    if ((itest % 16) != 0)
        fprintf(stderr, "\n");
     }
     else {
    while (itest < reclen) {
        fprintf(stderr, "%c", inpbuf[itest]);
        if ((++itest % 80) == 0)
       fprintf(stderr, "\n");
    }
```

```
    if ((itest % 80) != 0)
        fprintf(stderr, "\n");
     }

}

void disconnect()
{
     cmdeal(conv_id, &retcode);
     fprintf(stderr, "\ncmdeal done conv_id=%s retcode=%d\n", conv_id,
retcode);
}

int connect()
{
     cminit(conv_id, node, &retcode);
     fprintf(stderr, "\ncminit done conv_id=%s retcode=%d", conv_id, retcode);
     if (retcode) {
    disconnect();
    return (-1);
     }

     cmallc(conv_id, &retcode);
     fprintf(stderr, "\ncmallc done conv_id=%s retcode=%d", conv_id, retcode);
     if (retcode) {
    disconnect();
    return (-1);
     }

     if (rcv_state_after_allc) {
    cmrcv(conv_id, inpbuf, &inplen, &datarec, &reclen, &status, &rts,
          &retcode);
    fprintf(stderr, "\ncmrcv  done conv_id=%s retcode=%d", conv_id,
       retcode);
    display_input(0, inpbuf, reclen, status);
    if (retcode) {
        disconnect();
        return (-1);
    }
     }
     return (0);
}

int send_wait_for_answer(msg)
char *msg;
{
     strcpy(outbuf, msg);
     sndlen = strlen(msg);
     cmsend(conv_id, outbuf, &sndlen, &rts, &retcode);
     fprintf(stderr, "\ncmsend done conv_id=%s retcode=%d", conv_id, retcode);
     cmrcv(conv_id, inpbuf, &inplen, &datarec, &reclen, &status, &rts,
&retcode);
     fprintf(stderr, "\ncmrcv  done conv_id=%s retcode=%d", conv_id, retcode);
     display_input(0, inpbuf, reclen, status);
     if (retcode) { disconnect(); return (-1); }
     return (0);
}

int main()
{
     if (connect())
    return 1;
     if (send_wait_for_answer("account@Tpassword@Tuserid@E"))
    return 1;
```

```
    if (send_wait_for_answer("LWD@E"))      /*List working directory */
    return 1;
    if (send_wait_for_answer("cata,s,a@E")) /*List files */
    return 1;
    disconnect();
    return 0;
}
```

SimpleCpicCGI.pl: Simple procedural Perl/CGI connection to GCOS 8 host

```
#                              -*- Mode: Perl -*-
# SimpleCpicCGI.pl --- Simple procedural Perl/CGI example
# Author          : bard@gar.no
# Created On      : Mon Jun 26 09:48:13 2000
# Last Modified By: bard , bard@gar.no
#
#!/usr/local/bin/Perl
#
# Search the current directory for Perl modules
#
BEGIN {unshift(@INC,".")};
#
# load the Cpic7800 Perl module
#
use Cpic7800;
#
# Assign distributor country and number to 1st and 2nd command line args
respectively
#
$countryCode = $ARGV[0];
$rank        = $ARGV[1];
#
# Provide full country names for the HEML output
#
SWITCH:{
  $country = "OTHER",                    last SWITCH if ($countryCode eq
"OTH");
  $country = "OTHER",                    last SWITCH if ($countryCode eq
"oth");
  $country = "Australia",                last SWITCH if ($countryCode eq
"AUS");
  $country = "Australia",                last SWITCH if ($countryCode eq
"aus");
  $country = "Austria",                  last SWITCH if ($countryCode eq
"A");
  $country = "Austria",                  last SWITCH if ($countryCode eq
"a");
  $country = "Belgium",                  last SWITCH if ($countryCode eq
"B");
  $country = "Belgium",                  last SWITCH if ($countryCode eq
"b");
  $country = "OTHER",                    last SWITCH if ($countryCode eq
"OTH");
  $country = "OTHER",                    last SWITCH if ($countryCode eq
"oth");
  $country = "Denmark",                  last SWITCH if ($countryCode eq
"DK");
```

```
  $country = "Denmark",                  last SWITCH if ($countryCode eq
"dk");
  $country = "Finland",                  last SWITCH if ($countryCode eq
"FIN");
  $country = "Finland",                  last SWITCH if ($countryCode eq
"fin");
  $country = "France",                   last SWITCH if ($countryCode eq
"F");
  $country = "France",                   last SWITCH if ($countryCode eq
"f");
  $country = "Germany",                  last SWITCH if ($countryCode eq
"D");
  $country = "Germany",                  last SWITCH if ($countryCode eq
"d");
  $country = "OTHER",                    last SWITCH if ($countryCode eq
"OTH");
  $country = "OTHER",                    last SWITCH if ($countryCode eq
"oth");
  $country = "Italy",                    last SWITCH if ($countryCode eq
"IT");
  $country = "Italy",                    last SWITCH if ($countryCode eq
"it");
  $country = "Netherlands",              last SWITCH if ($countryCode eq
"NL");
  $country = "Netherlands",              last SWITCH if ($countryCode eq
"nl");
  $country = "Norway",                   last SWITCH if ($countryCode eq
"N");
  $country = "Norway",                   last SWITCH if ($countryCode eq
"n");
  $country = "OTHER",                    last SWITCH if ($countryCode eq
"OTH");
  $country = "OTHER",                    last SWITCH if ($countryCode eq
"oth");
  $country = "Spain",                    last SWITCH if ($countryCode eq
"ES");
  $country = "Spain",                    last SWITCH if ($countryCode eq
"es");
  $country = "Sweden",                   last SWITCH if ($countryCode eq
"S");
  $country = "Sweden",                   last SWITCH if ($countryCode eq
"s");
  $country = "Switzerland",              last SWITCH if ($countryCode eq
"CH");
  $country = "Switzerland",              last SWITCH if ($countryCode eq
"ch");
  $country = "United Kingdom",           last SWITCH if ($countryCode eq
"UK");
  $country = "United Kingdom",           last SWITCH if ($countryCode eq
"uk");
  $country = "United States of America", last SWITCH if ($countryCode eq
"USA");
  $country = "United States of America", last SWITCH if ($countryCode eq
"usa");
  $nothing = 1;
}
#
# The "tp8test" host must be defined in your cpic.cfg configuration file
#
$host = "tp8test";
#
# Prepare a conversation
#
($cid, $rc) = Cpic7800::Initialize_Conversation ($host);
```

```
#
# Establish connection
#
$rc = Cpic7800::Allocate ($cid);
die if ($rc);
#
# Receive the first form
#
($buffer, $dr, $status, $rts, $rc) = Cpic7800::Receive ($cid);
die if ($rc);
#
# Emulate the keystrokes "1" and "Transmit"
#
$rc = Cpic7800::Send ($cid, "1\@E1\@E".$countryCode."\@E".$rank."\@E", 0,
$rts);
die if ($rc);
#
# Receive the next form
#
($buffer, $dr, $status, $rts, $rc) = Cpic7800::Receive ($cid);
#
# assign company name, distributor name and distributor e-mail
#
$company     = trim(substr($buffer,740,30));
$distributor = trim(substr($buffer,1300,30));
$Email       = trim(substr($buffer,1380,30));
#
# Generate the HTML output to send back to client
#
print "Content-type: text/html","\n\n";
print "<html>", "\n";
print "<body>", "\n";
print "You requested contact information on a <b>G & R</b> distributor in
\"".$country."\".<p>\n";
print "We found <b>".$distributor."</b> of <b>".$company."</b>.<p>\n";
print "You may send <b>".$distributor."</b> an e-mail by clicking: \n";
print "<a href=\"mailto:".$Email."\"><b>".$Email."</b></a>","\n";

print "</body>","\n";
print "</html>","\n";
#
# Deallocate the session
#
Cpic7800::Deallocate ($cid);

exit (0);

#
# method to strip trailing blanks
#
sub trim{
  my @out = @_;
  for(@out){
    s|\s+$||;
  }
  return wantarray ? @out : $out[0];
}
```

FancyCpicCGI.pl: Simple OO Perl/CGI connection to GCOS 8 host

```
#                               -*- Mode: Perl -*-
# FancyCpicCGI.pl --- Simple OO Perl/CGI example
# Author          : bard@gar.no
# Created On      : Mon Jun 26 10:32:03 2000
# Last Modified By: bard , bard@gar.no

#!/usr/local/bin/Perl
#
# Search the current directory for Perl modules
#
BEGIN {unshift(@INC,".")};
#
# load the Cpic7800 Perl module
#
use Cpic7800;
#
# load the CGI Perl module
#
use CGI ':standard';

#
# The "tp8test" host must be defined in your cpic.cfg configuration file
#
$host = "tp8test";
#
# Generate the form and instantiate CGI objects (save state, parse form
request etc.)
#
print header;
print start_html(
        -title      => 'Perl GLAPI SDK Example',
        -author     => 'bard@gar.no',
        -meta       =>{'keywords'=>'Perl GlAPI CPI-C 7800',
              'copyright'=>'copyright 2000 G & R'},
        -text       => '#000000',
        -bgcolor    => '#ffffff',
        -link       => '#0000ee',
        -vlink      => '#551a8b',
        -alink      => '#ff0000'
       ),
  img(
      {
       src    => 'http://www.jabberwocky.org/~bard/images/glapirtcpic.gif',
       border => '0',
       height => '148',
       width  => '350',
       align  => 'bottom'
      }),
  b(i('CGI Perl Example')),

  p,
  start_form,
  "Which country? ",
  popup_menu(
        -name=>'country',
        -values=>['Norway',
              'Australia',
```

```
                'Austria',
                'Belgium',
                'Denmark',
                'Finland',
                'France',
                'Germany',
                'Italy',
                'Netherlands',
                'Other',
                'Spain',
                'Sweden',
                'Switzerland',
                'United Kingdom',
                'United States of America'
               ]),
    p,
    "Which distributor? ",textfield(
                 -name => 'distributorNumber',
                 -size => '2',
                 -value => '1'
                ),
    p,
    submit,
    end_form,
    hr;
#
# Prepare a conversation
#
($cid, $rc) = Cpic7800::Initialize_Conversation ($host);
#
# Establish connection
#
$rc = Cpic7800::Allocate ($cid);
die if ($rc);
#
# Receive the first form
#
($buffer, $dr, $status, $rts, $rc) = Cpic7800::Receive ($cid);
die if ($rc);
#
# Emulate a sequence of keystrokes
#
$rc = Cpic7800::Send ($cid,
"1\@E1\@E".encodeCountry(param('country'))."\@E".param('distributorNumber')."
\@E", 0, $rts);
die if ($rc);
#
#Receive the next form
#
($buffer, $dr, $status, $rts, $rc) = Cpic7800::Receive ($cid);

$company     = trim(substr($buffer,740,30));
$distributor = trim(substr($buffer,1300,30));
$Email       = trim(substr($buffer,1380,30));
#
# Deallocate the session
#
Cpic7800::Deallocate ($cid);


if (param()) {

  print
```

```
    "You requested contact information on a ",b("G & R")," distributor in
",em(param('country')),".",
    p,
    "We found ".b($distributor)." of ".b($company),
    p,
    "You may send ".b($distributor)." an e-mail by clicking:
".a({href=>"mailto:".$Email},b($Email)),
    p,
      hr;


}
print end_html;

#
# misc methods
#
# trim off trailing white space
#
sub trim{
  my @out = @_;
  for(@out){
    s|\s+$||;
  }
  return wantarray ? @out : $out[0];
}
#
# Covert country names to country codes recognized by the host application
#
sub encodeCountry{
  my $country = shift;

 SWITCH:{
    $countryCode = "AUS", last SWITCH if ($country eq "Australia");
    $countryCode = "A",   last SWITCH if ($country eq "Austria");
    $countryCode = "B",   last SWITCH if ($country eq "Belgium");
    $countryCode = "DK",  last SWITCH if ($country eq "Denmark");
    $countryCode = "FIN", last SWITCH if ($country eq "Finland");
    $countryCode = "F",   last SWITCH if ($country eq "France");
    $countryCode = "D",   last SWITCH if ($country eq "Germany");
    $countryCode = "IT",  last SWITCH if ($country eq "Italy");
    $countryCode = "NL",  last SWITCH if ($country eq "Netherlands");
    $countryCode = "N",   last SWITCH if ($country eq "Norway");
    $countryCode = "OTH", last SWITCH if ($country eq "Other");
    $countryCode = "ES",  last SWITCH if ($country eq "Spain");
    $countryCode = "S",   last SWITCH if ($country eq "Sweden");
    $countryCode = "CH",  last SWITCH if ($country eq "Switzerland");
    $countryCode = "UK",  last SWITCH if ($country eq "United Kingdom");
    $countryCode = "USA", last SWITCH if ($country eq "United States of
America");
    $nothing = 1;
  }
  return $countryCode;
}
```

Appendix: Host Links Manuals

Below you find a complete list of all available Host Links manuals:

Installation	
Host Links Servers	Installation and Configuration on UNIX/Linux
Host Links Emulators	Installation and Configuration on UNIX/Linux
Host Links	Installation and Configuration on Windows
Line handling	
Gline	Line Handler and DSA/OSI Configuration
Ggate	Transparent Gateway
Gproxy	Network Manager & SNMP Proxy Agent
G&R SSL	Using SSL for security in G&R products
GlAPI	Application Programming Interfaces
Emulations	
Gspool	Network Printer Emulation
GUFT	Unified File Transfer
G3270	Emulating IBM 3270 Terminals
G5250	Emulating IBM 5250 Terminals
Pthru	Gateway to the Bull Primary Network
Qsim	Emulating Questar DKU7107-7211 & VIP7700-7760
V78sim	Emulating VIP7801 & VIP7814
Gweb	Web Browser Front-end for DKU, VIP7700-7760, VIP7800, IBM3270 and IBM5250 Emulations

Appendix: Host Links Server Administration

Gmanager is the Host Links administration tool. It can be used to control, configure and monitor all the G&R Host Links server programs.

The dialog and interaction between the server programs and Gmanager is based on information located in a database file `_active.srv` that is located in the Host Links `servers` directory. The first time a Host Links server program starts up it registers itself in this 'active' file. Thereafter the server program updates this database with status information whenever the server is active.

The Gmanager program is available in 2 different versions – a Windows GUI based version `gmanw.exe` and a character based subset `gman` (UNIX/Linux binary) or `gman.exe` (PC console application).

The basic functionality of the two versions is the same, but the Windows version interfaces directly to other Windows-only Host Links administrative tools (*`Gconfig`*, *`Gservice`*), and can also start the browser directly to view HTML reports produced by Gproxy, if enabled, or to view the HTML pages associated with a *`Gweb`* or *`Glink for Java`* installation.

The *`Gproxy`* reports, *`Gweb`* and *`Glink for Java`* web pages are of course available to administrators of UNIX/Linux Host Links systems, and can be viewed by starting a browser manually, and connecting to the appropriate URLs:

```
http://mysite.mydomain.com/Gproxy
http://mysite.mydomain.com/Gweb
http://mysite.mydomain.com/GlinkJ
```

A summary of the available functions follows. The Windows-only functions are marked.

Gmanager can be used to perform the most common Host Links administrative tasks i.e.:

- View the last reported status information from the servers
- Stop or restart all servers, start a new server, stop, restart or delete a server
- Send a command to a server
- Load the DSA configuration into an editor, compile the DSA configuration
- Start the server configuration program or the configuration wizard (Windows)
- Load the `Gservice` configuration into an editor (Windows)
- Edit the product specific configuration files
- View a server log file, a server trace file or the server configuration file
- View program version numbers, program link information (Windows)
- View license info and license usage (Windows)
- View Host Links environment information, the 'VMAP' (Windows)
- Start Gdir directory administrator, Ggate monitor, Gspool monitor
- Gather all traces and logs for trouble-shooting by support
- Gping a DSA node to check the connection, use Gerror to explain error code
- Set a transport route state (down, enbl, lock, used)
- Check if a printer is on-line, request a list of bins
- Connect directly to the `Gproxy`, `Gweb`, `GlinkJ` HTML pages(Windows)

The commands that are accepted by all servers are:

- DOWN - terminates the server
- STATUS - reports server-specific status information to the log file
- PARAM - allows the operator to give a command line parameter to the server. Note that some parameters do not work when given interactively i.e. they can only be handled at server startup time
- DEBUG ON/OFF - toggles on and off tracing interactively

Additionally, the server in question might support other interactive commands. For a description of the supported commands, check the server-specific documentation.

Appendix: G&R/DSA utilities

The Gline package includes a set of Gline communication utilities. These are used when testing and debugging connection problems. The utilities are delivered as part of the Gline package and can be used without any additional configuration. The nodes to be tested must of course be configured in the `dsa.cfg` file.

Gconame

Lists the parameters generated from a given CONAME. The utility works for both CONAME and RESOURCE e.g.:

```
gconame tnviptm

Checking 'dsa.cfg' for coname 'tnviptm'
Coname: tnviptm, type TM, parameters:
-DA misfld
-S_
-D_
-CODE 0000
-CODE 1000
-CODE 1800
-TEXT Remote SCID?:
-CODE 4700
-TEXT Remote application?:
-CODE 1400
-CODE 1600
-TEXT Password?:
```

Gerror

Shows the text message associated with a DSA reason code. Only the most common codes are supported i.e. the ones related to network, transport and session communication layers. Errors generated by the OSI-stack on the Host Links platform are not covered by this utility; please refer to the documentation from the vendor of the stack e.g.:

```
gerror 0109
Reporting component: Session control (01) 0109, Dialog
protocol error or negotiation failed (wrong logical record).
```

For a detailed description of all reason codes, please consult the Bull manual ***OSI/DSA Network System Messages and Return codes*** (39A2 26DM).

Glnode

List and verify the communications parameters of the local node e.g.:

```
glnode
Local node name : GRDL
Local session control id : GRDL
DSA200 address (area:tsm): 54:60 (36:3C)
```

Gmacfix

When you connect to FCP cards on Bull mainframes via an Ethernet port on the LAN-Extender the mainframe address is given in Ethernet (LLC) format. If you connect to an FDDI adapter you must convert the MAC address to SMT. e.g.:

```
gmacfix 080038000fab
MAC address 080038000fab = 10001c00f0d5
```

Gping

Connects to a remote system using the Gline parameters set on the command line. If successful it returns ‘connected to application’, otherwise it shows the error code returned e.g.:

```
gping -li dsa -dn b7dl -da iof -du jim -pw mydogsname
Gping - $$DSA: Connected to application
```

Grnode

Return the parameters (in `dsa.cfg`) and the state of a remote node e.g.:

```
grnode b6dl
Checking 'dsa.cfg' for node 'b6dl'
Session control id : B6DL
DSA200 address (area:tsm) : 1:5 (1:5)
Inactivity interval : 0
Route 0
Load balance percentage : 0
TP class : 2
TP expedited : 0
TPDU size : 0
Network address : 130405
```

Gtrace

Same as `gping` but writes the DSA communication trace on the user's terminal (applicable to UNIX versions) e.g.:

```
gtrace -li dsa -dn ln40 -da snm151
D6:Application event @ 14:17:17.6003. tokenitem = 00
D6:Application event @ 14:17:17.6082. tokenitem = 00
D6:Connect request called, node = LN40
D6:OurBufferSizes. ApplMaxXmit = 511, ApplMaxRecv = 500
Rec:4000 0002 s:2
Rec:506B 0010 s:16
etc etc
Gtrace - line trace ending.
Gtrace - $$DSA: Connected to application.
```

Gtsupd

Update the state of a transport route. Transport routes can be set automatically in a disabled state if a backup route is configured. When such a state change occurs the route will be set back to the enabled state after a configurable timer has expired. The default is 15 minutes. You can reset the state of such a route with the `gtsupd` utility e.g.:

```
gtsupd hipp -st enbl
TS-entry 'hipp' updated OK. Old state = LOCK, new state =
ENBL
```

Appendix: Host Links trace

If you experience any kind of problem when using a Host Links application, the application trace file and/or the line handler trace file will provide useful documentation of the problem.

Trace activation

The Host Links products automatically create sub-directories in the debug directory when debug is activated: at product level using the `-dbg` parameter, or at line level using the `-d_` or `-s_` parameters to the line module.

Windows server	`gspool -id gs1 -dbg -ps \\SERVER\LEXMARK -li dsa -da tptst -d_ on`
UNIX Linux	`gspool -id gs1 -dbg-pc lp -li dsa -da tptst -d_ on`

Most G&R products include a facility for setting product or line parameters dynamically. It is therefore generally possible to turn on debug or trace without modifying the command line or configuration of a production system.

Trace types

All Host Links products accept a parameter `-dbg`, which starts an application level trace of internal events. This is useful when investigating malfunctions or looking closely at product behaviour.

All Gline line handlers accept a parameter `-d_` to turn on a data trace. It records data and enclosure level being exchanged with the line handler. It is useful when documenting product malfunction e.g. an emulation error, because it records exactly what the host sends and what the G&R application replies. It can be used to simulate a customer situation, reproduce a problem and to verify that a correction fixes the documented problem.

All Gline line handlers accept a parameter `-s_` to turn on a session trace. It records the raw data being exchanged between the line module and the underlying transport layer (e.g. OSI Transport, or TCP socket), as well as internal events and protocol states. It is useful when investigating protocol failures such as unsuccessful connect attempts or abnormal disconnections.

Structure

The Host Links file structure includes a debug directory to collect the trace and debug files in one location where the permissions can be adjusted as required for security. By default only the Host Links administrator can access the directory. The debug directory is created by the initialization procedure and located (by default) in:

Windows server	`\gar\debug`
UNIX Linux	`/usr/gar/debug`

If the application is a client type of application, a debug sub-directory with the same name as the user (UNIX username or PC login name) is created and all debug files are located there. This includes the line level trace except in the special case where the client application connects via Ggate and the line level trace is written on the Ggate system using the Ggate DSA node name as a debug sub-directory.

If the application is a server type of application, then a sub-directory will be created using the DSA node name on behalf of which the server application is executing. If the server does not use DSA the default local session control name is still used if there is a `dsa.cfg` file. If there is no `dsa.cfg` file then the system's UNIX or Windows communications node name is used. You can find this name using the command `uname -n` on UNIX systems, or the Network section of the control panel on Windows systems. This covers situations where several instances of a server are executing on the same system and accepting incoming calls to different DSA node names, or where several Host Links systems using the same server product share a file system.

Tracing Ggate

When Glink, a Host Links client or a customer application based on GlAPI connects through Ggate to the application, the line handler trace is generated on the Ggate system, with the name and location shown in the table:

Windows server	\gar\debug\NODE\gga**NN-PPPP**.dbg
UNIX Linux	/usr/gar/debug/NODE/gga**NN-PPPP**.dbg

NODE is the local DSA node name used by the Ggate system.

The trace file name consists of the prefix `ggaNN-` followed by the IP-address of the client, suffixed by `.dbg` for a terminal session or `-dbp` for a printer session. The following is a trace file name for Ggate session sequence number 5 executing on Host Links system `GRDL` initiated from a Glink client on IP-address `jim.gar.no`:

```
gga05-jim.gar.no.dbg
```

This file, and possibly also a Glink debug file and a Glink communication trace file activated by the `/J` command line parameter will be needed by the support engineer investigating any problem.

To enable a line handler trace through Ggate the product's start-up command or configuration file would look like this:

```
-LI YYY:ZZZZ -S_ -D_
```

(YYY =line handler identification, i.e. DSA or DIWS)
(ZZZZ =IP-address of the system running Ggate)

Examples - G&R products

Examples of directory and file names in the debug structure are:

/usr/gar/debug/jim	**Debug directory for user 'jim'**	
qsm.dbg	Qsim emulator debug file	-dbg

qsm-gli.dbg	Qsim host line trace	-li dsa -s_
pth-glit.dbg	Pthru terminal line trace	-term -s_
pth-glih.dbg	Pthru -host line trace	-li dsa -s_
g32.dbg	G3270 emulator debug file	-dbg
g32-gli.dbg	G3270 host line trace	-s_
/usr/gar/debug/mike	**Debug directory for user 'mike'**	
v78.dbg	V78sim emulator debug file	-dbg
v78-gli.dbg	V78sim host line trace	-li dsa -s_
guf.dbg	GUFT client debug file	-dbg
guf-gli.dbg	GUFT client host line trace	-li dsa -s_
/usr/gar/debug/en01	**Debug directory for node 'en01'**	
guf.def	GUFT server debug file	-dbg
guf-gli.def	GUFT server host line trace	-li dsa -s_
gli-gli.dsa	DSA listener host line trace	-s_
gli-gli.diw	DIWS listener host line trace	-s_
gsp.def	Gspool (default -id) debug file	-dbg
gsp-gli.def	Gspool (default -id) host trace	-li dsa -s_
gga01-mike.gar.no.dbg	Ggate line trace, first Glink	-s_
gga02-mike.gar.no.dbg	Ggate line trace second Glink	-s_
/usr/gar/debug/en02	**Debug directory for node 'en02'**	
gsp.abc	Gspool (-id abc) debug file	-dbg
gsp-gli.abc	Gspool (-id abc) host trace	-li dsa -s_
gspc-gli.def	Gspool DPF8 command trace	-li tcp -s_
gspd-gli.def	Gspool DPS8 data trace	-li tcp -s_

gsp._00	Gspool started on demand debug	-dbg
gsp-gli._00	Gspool started on demand trace	-li dsa -s_

CPI-C and Gweb trace files

Gweb uses the CPI-C libraries so the Gweb debug structure is exactly the same as for CPI-C, except that Gweb inserts its own product identifier into the file name structure. CPI-C applications use the 'client' style of debug and create a debug directory with the UNIX username or PC login name used by the process that started them.

The application level debug (-dbg) and line trace (-s_ and -d_) are set in the cpic.cfg file. The line trace goes to the debug directory, with the name built up as follows:

```
<product_id><session_id>-<process_id>.<debug_type>
```

product_id	*Value*	*Comment*
	cpi	CPI-C API
	cp3	CPI-C 3270
	cp7	CPI-C 7800
	cpd	CPI-C DKU
	gw3	Gweb3270
	gw7	Gweb7800
	gwd	Gwebdku
session_id	(nn)	If multi-session application, 1-63
process_id	n (n n n...)	Varies by platform
debug_type	dgb	Application level debug
	gli	Line trace

Example:

`\gar\debug\system`	debug directory for user "`system`"	
`cpi-16.dbg`	CPI-C single session debug	`-dbg`
`cpi-16.gli`	CPI-C single session line trace	`-li dsa -s_`
`cpi2-123.dbg`	CPI-C session 2 application debug	`-dbg`
`gw7-20172.gli`	Gweb7800 host line trace	`-li dsa -s_`

Appendix: OSI/DSA Return Codes and Error Messages

OSI/DSA error codes

Below is a list of OSI/DSA error codes and the corresponding description. These are the same descriptions that the `G&R/Gerror` utility will display when given the DSA code as a parameter.

code	Description
00xx	**General Errors**
0001	Open Failure in LC - Reject for unknown reason
0002	Open Failure in LC - Acceptor customer node inoperable
0003	Open Failure in LC - Acceptor customer node saturated.
0004	Open Failure in LC - Acceptor mailbox unknown.
0005	Open Failure in LC - Acceptor mailbox inoperable.
0006	Open Failure in LC - Acceptor mailbox saturated.
0007	Open Failure in LC - Acceptor application program saturated
0008	Connection refused. Transport protocol error or negotiation failed.
0009	Open Failure in LC - Dialog protocol error or negotiation failed
000A	Open Failure in LC - Presentation protocol error or negotiation failed
000B	Open Failure in LC / Connection refused lack of system resources
000C	Open Failure in LC / Connection refused from GCOS7 duplicate user
000D	Open Failure in LC, Duplicate implicit LID / Q class not started
000E	Open Failure in LC, Duplicate GRTS Id / lack of memory resources
000F	Open Failure in LC, No Logical line declared for DACQ / 7 connection refused
0010	Open Failure in LC, GCOS 8 GW Missing translation / Incorrect device length in ILCRL.
0011	Open Failure in LC, DAC connection not initialized / Too many jobs executing

0012	Open Failure in LC, No binary transfer / impossible to start the IOF job
0013	Open Failure in LC, connection is not negotiated in FD mode / impossible to start the IOF job
0014	Disconnection - Timeout resulting from absence of traffic.
0016	Option missing for an RBF mailbox.
0017	Connection refused - Incorrect access right for MB.
0018	Connection refused - Incorrect access rights for the application.
0019	Connection refused - Unknown pre-negotiated message path
001A	Connection refused - Security validation failed.
001B	Connection refused - Unknown acceptor mailbox extension.
001C	Connection refused - Inoperable acceptor mailbox extension.
001D	Connection refused - Invalid Message group number.
001F	Disconnection - no more memory space.
0020	Connection refused - Unknown node.
0021	Connection refused - inaccessible node or Host down.
0022	Connection refused - saturated site.
0023	Connection refused - inoperable mailbox.
0024	(X.25) Packet too long. Problem with packet size. / Connection block already used.
0030	Syntax Error - option not known (received on close VC).
0031	(X.25) No response to call request packet - timer expired.
0033	(X.25) Timer expired for reset or clear indication.
0039	Disconnection - transport protocol error (MUX).
003C	Presentation Control Protocol Error
003E	The application has not the turn
003F	Message group closed
0040	(X.25) Facility code not allowed. / Connection refused - unknown node
0041	Connection refused - path not available.
0042	Connection refused - Duplicate USER ID / Facility parameter not allowed
0044	(X.25) Invalid calling address.
0045	(X.25) Invalid facility length.
0047	(X.25) No logical channel available.
004F	DNSC: (X.25) Invalid call packet length.
0050	Normal disconnection (GCOS3/8)
0051	Error or Event on LC initiated by GW
0052	Error or Event on LC initiated by GW.
0053	Error or Event on LC initiated by GW. TCall
0054	Error or Event on LC initiated by GW. DIA in LOCK State
0055	Error or Event on LC initiated by GW. DIA error

0056	Error or Event on LC initiated by GW. GW has no known explanation.
0057	Error or Event on LC initiated by GW. Reject mailbox permanent
0058	Error or Event on LC initiated by GW. No more input lines in DACQ
0059	Time-out on GCOS 3/8 gateway.
005A	Error or Event on LC initiated by GW. Disconnect from terminal without reason
005B	Error or Event on LC initiated by GW. Wrong letter or wrong record
005C	Error or Event on LC initiated by GW. Forbidden letter received
005D	Error or Event on LC initiated by GW. Forbidden letter received
005E	Error or Event on LC initiated by GW. No buffer for secondary letter
005F	Error or Event on LC initiated by GW. No buffer for fragmented letter
0060	Error or Event on LC initiated by GW. Disconnect on end of phase record
0061	Error or event on LC initiated by GW. No buffer for control letter.
0062	Error or event on LC initiated by GW. Mailbox in closing phase
0064	Error or event on LC initiated by GW. Flow control error.
0065	Error or event on LC initiated by GW. CH locked by operator.
0066	Error or event on LC initiated by GW. Disconnect with a normal TMG F2 exchange.
0067	Error or event on LC initiated by GW. Teletel rerouting error from DACQ
0068	Error or event on LC initiated by GW. Teletel routing error from DACQ
0069	Error or event on LC initiated by GW. Teletel rerouting error from TM
006A	Error or event on LC initiated by GW. Teletel rerouting error from TM
006B	Syntax error - text too long.
006C	Syntax error - illegal object in a GA command.
006D	Syntax error - unknown node Id.
0078	Syntax error - illegal command for this object.
0079	Syntax error - illegal date.
007F	(X.25) No route available for X.25 switching.
0081	No more network routes available for switching.
0082	(X.25) Hop count reached for X.25 switching.
0083	(X.25) Flow control negotiation error.
0085	(X.25) Frame level disconnection.
0086	(X.25) Frame level connection.
0087	(X.25) Frame level reset.

0090	Frame level not set.
0092	(X.25) X.25 Echo service in use.
0093	(X.25) Incorrect password for PAD connection.
0094	(X.25) No more PAD connections allowed.
0096	(X.25) TS SX25 or NU X25 objects locked.
009C	(X.25) Invalid packet header. X.25 protocol error.
009D	(X.25) Incompatible header. X.25 protocol error.
009E	(X.25) Logical Channel Number too high.
009F	(X.25) Incorrect packet type.
00B2	Use of invalid password through PAD
00B6	Unknown mailbox selection for PAD connection using the PAD password.
00C0	(X.25) Normal disconnection.
00D7	(X.25) TS image (of type DSA or DIWS) in LOCK state.
00DE	(X.25) NS RMT or NR SW in LOCK state.
00E1	Connection refused. Mailbox is not in ENBL state.
00E6	QOS not available permanently.
01xx	**Session Control**
0100	Logical connection accepted or normal termination
0101	Rejection for unknown reason or abnormal termination
0102	Acceptor node inoperable.
0103	Acceptor node saturated. When a node has no available resources
0104	Acceptor mailbox unknown.
0105	Acceptor mailbox inoperable.
0106	DNS: Acceptor mailbox saturated.
0107	DNS: Acceptor application program saturated.
0108	Transport protocol error or negotiation failed (DSA 200 only).
0109	Dialog protocol error or negotiation failed. (Wrong logical record).
010A	Time-out on session initiation / unknown LID
010B	Acceptor mailbox extension unknown.
010C	Acceptor mailbox extension inoperable.
010D	Invalid Session Number.
010E	Unknown node.
010F	System error. System generation error or insufficient memory space
0110	Application abnormal termination. Subsequent to an abnormal occurrence in the dialogue
0111	Normal terminate rejected.
0112	Protocol not supported.
0113	Session control service purged by user.
0115	Disconnection Time-out on message group initiation.
0117	Incorrect Access Right for MB
0118	Incorrect Access Right for the Application

0119	Pre-negotiated Message Path Descriptor unknown
011A	Security validation failed
011E	Incorrect object status
011F	Not enough memory space available.
0120	Node unknown.
0121	The channel object (CH) is in LOCK state
0122	Saturation - no plug available
0123	Object status = LOCK
0124	Connection block (TSCNX) already used
0125	Disconnection already running
0126	The connection block (TSCNX) is disconnected (or not connected)
0127	Change Credit value < 0
0128	Ineffective Change Credit (delta = 0)
0129	No more deferred letters
012B	"Reinitialization" Request
012C	"Reinitialization" in progress
012D	"Reinitialization" in progress, letters are dropped
012E	Close virtual circuit. Either no mapping exists between PA/NR or CL and VC/NS
012F	Null connection object index.
0130	Undefined function at Sysgen time.
0131	Letter too large with respect to the negotiated size.
0132	The received letter is longer than the size which was
0133	Disconnection of the session control user
0134	Interface error on EOR (End-Of-Record) processing.
013C	Presentation control protocol error.
013E	You do not have the turn.
013F	Message group closed.
0140	Session is closed.
0151	Request refused, no system buffers available.
0152	Incorrect addressing record.
0153	No presentation record in the ILCAL or ILCRL
0154	Negotiation failed on session mode
0156	Negotiation failed on resynchronization.
0157	Negotiation failed on END to END ACK
0158	No presentation record in the connection letter
0159	Negotiation failed on session mode
015A	Negotiation failed on letter size (in the Logical Connection record).
015B	Negotiation failed on resynchronization (in the Logical Connection record).
015C	Negotiation failed on end-to-end ACK (Logical Connection record).

015D	No support of the "letter" interface because Multirecord is not negotiated.
0160	Incorrect TSPACNX table.
0161	Protocol error on letter reception.
0162	Negotiation failure.
0163	Record header length error.
0164	Protocol error.
0165	Protocol error reception of control letter.
0166	Type or length error on interrupt letter.
0167	Protocol error on reception of data letter.
0168	Dialog protocol error.
0169	Unknown event.
016A	Protocol error on data transfer.
016B	Invalid status for a disconnection request.
016C	Invalid status for a recover
016D	Invalid status for a suspend/resume request.
016E	Negotiation failure.
016F	Unknown command.
0170	Error in presentation protocol
0171	Letter header length error in
0172	ILCAL is not DSA 200 protocol.
0173	Error in session record.
0174	Normal disconnection, without complementary reason code.
0175	Letter is not in ASCII or EBCD.
0176	Connection protocol letter header
0177	Letter header protocol error.
0178	Record header protocol error.
0179	Record header length error.
017A	Mbx record header length error.
017B	Error on buffer transfer.
017C	DSA 200 record header protocol
017D	DSA 300 record header protocol
017E	Unsupported connection options.
017F	Character error in ASCII string.
0180	No segmented record size.
0181	Invalid mailbox object index.
0182	Mapping error for a remote connection.
0190	No more buffers.
0191	Byte count is greater than GP.
0192	Byte count is greater than GP.
0193	Byte count is greater than GP.
0194	Byte count is greater than GP.

0195	Byte count is greater than GP.
0196	Byte count is greater than GP.
0197	Byte count is greater than GP.
0198	No more buffers.
0199	Byte count is greater than GP.
019A	Byte count is greater than GP.
019B	Byte count is greater than GP.
019C	Byte count is greater than GP.
019D	Byte count is greater than GP.
019E	Byte count is greater than GP.
019F	Byte count is greater than GP.
01A0	Invalid transfer state.
01A1	Suspend protocol running.
01A2	Suspend protocol running.
01A3	Recover protocol running.
01A4	Forbidden function in write request. ($WRITE)
01A5	Conflicting parameters for segmented record. (SWBREC)
01A6	Protocol conflict - suspend/recover.
01A7	Protocol not supported - letter/end-to-end ACK. (SWBLET)
01A8	Multi-record letter in progress.
01A9	Interrupt request forbidden.
01AA	Send control record request forbidden. (SCTROL)
01AB	Forbidden for TWA session - turn is here. (SREAD)
01AC	Termination forbidden - suspend or recover in progress. (STERM)
01C0	No space available for downstream connection request. (SMECNX)
01C1	No space available for upstream connection request. (SMUCNX)
01C2	No space available for upstream SCF connection. (SMRCNX)
01C3	No space available for session context. ($SCTX)
01E0	Enclosure or data length error for a write request. ($WRITE)
01E1	Enclosure or data length error for a write segment record request. (SWBREC)
01E2	Enclosure error for 'give turn' request. (SGVTRN)
01E3	Interrupt request is not demand turn, attention/data attention, or purge record.
01E4	Input status for a send control letter is not permitted.
01E8	Write request without turn.
01E9	Write segmented record request without turn.
01EA	Write segmented letter request without turn.
01EB	Send control letter request without turn.
01EC	Disconnection request without turn.
02xx	**Presentation Control**
0201	Protocol level not supported

0202	Application designation protocol error.
0203	Character encoding error. TM cannot support the proposed encoding.
0204	Character set error. TM cannot support the proposed character set.
0205	Character subset error. TM cannot support the proposed character subset.
0206	Incorrect record encoding.
0207	Incorrect parameter encoding.
0230	Data presentation control error. The presentation control proposed for this session cannot be used
0231	Device type is incompatible with the configuration.
0232	TM control protocol is incorrect.
0233	Device-sharing attributes are invalid.
0234	Initiator or acceptor configuration is not correct.
0235	Logical device index error.
0236	Number of logical devices is incompatible with the configuration.
0237	TM protocol record not supported.
03xx	**Terminal Management**
0300	Sysgen error WARNING. There is no mapped object; some objects will be spare.
0301	Operator requested session abort or logged.
0302	Idle time run out after secondary network failure.
0303	Idle time run out for no traffic.
0304	Form not found.
0305	Operator requested suspension.
0306	Destructive attention send on the session.
0307	Unknown TX addressed in this session. TM is unable to a the session.
030A	Protocol error. A record was received which did not comply with current standards
0310	Insufficient resources. The receiver cannot act on the request because of a temporary
031E	Incorrect value for Retry or Wait parameters on UP LL command.
0320	Function not supported.
0321	Parameter error. This can result
0322	Resource not available. The
0323	Intervention required (on principal device).
0324	Request not executable.
0325	EOI required.
0326	Presentation space altered, request executed.
0327	Presentation space altered, request not executed.
0328	Presentation space integrity lost.
0329	Device busy. The device is busy and cannot execute the request.

032A	Device disconnected.
032B	Resource not configured.
032C	Symbol set not loaded.
032D	Read partition state error.
032E	Page overflow.
0330	Subsidiary device temporarily not available.
0331	Intervention required at subsidiary device.
0332	Request not executable because of subsidiary device.
0340	TM cannot accept a new connection.
0341	Object status incorrect.
0342	The TM configuration is not correct.
0343	Unknown TX addressed on this session.
0344	Data presentation protocol error.
0345	Device type is incompatible with the configuration, or is not supported.
0346	TM control protocol incorrect.
0347	Device shareability attributes are invalid.
0348	Initiator or acceptor configuration is not correct.
0349	Logical device index error.
034A	Number of logical devices incompatible with the configuration.
0350	Disconnection of TM after reinitialization of the network.
0360	File not found. (Welcome and Broadcast Messages)
0361	Site not found. (Welcome and Broadcast Messages)
0362	NASF error. (Welcome and Broadcast Messages)
0370	No-session timeout. Device disconnected.
0371	No-input timeout. Device disconnected.
0372	No-output timeout. Device disconnected.
0373	Timeout due to no backup session being initiated.
0374	Timeout due to no backup session being established.
0375	Connection refused because of late activation of back up session.
0376	Disconnection of current session to switch to backup session.
0380	AUTOCN parameter not declared.
0381	Mixed ETB in data sent by VIP screen and cassette
0382	Data header sent by the terminal incorrect.
0383	Desynchronization in the exchange of data.
0384	KDS block count error.
038C	Remote terminal is not connected
0390	Unknown mailbox.
0391	No call packet to return.
0392	No "Possibility" command to return Protocol error
03C0	Slave device disconnection.

17xx	Network Layer
1701	PAD connection refused.
1702	Flow control error.
1706	Logical channel number not zero in restart packet.
1707	Illegal packet length or use of D-bit forbidden.
1708	Illegal header.
1709	Illegal Logical Channel Number.
1710	Invalid packet type for the automaton state. Protocol error
1711	Incorrect packet type.
1712	Inconsistent network parameters in the generation file.
1713	No more space.
1714	DSAC network layer object not usable.
1717	USED/ENBL transition. Transport station is locked.
1718	USED/ENBL transition. This is a back-up NR.
1719	USED/ENBL transition. Dynamic close due to load.
171A	USED/ENBL transition. Transfer time-out has elapsed.
171B	USED/ENBL transition. This is a back-up NR.
171C	USED/ENBL transition. Transport station is idle.
171E	USED/ENBL transition. NR object is locked.
171F	ENBL/LOCK transition. NR HDLC has no more memory space.
1721	Remote station is inaccessible via the configured network. Check
1723	Incorrect PAD password.
1724	Virtual circuit already in use. LCN (Logical Channel Number) too high.
1725	Invalid virtual circuit.
1726	Packet too short. Protocol error for the equipment directly connected to the Bull Datanet.
1727	Incompatibility between the generation parameters of two communicating systems on window or packet size.
1729	Packet size in communicating systems not the same.
1731	Timer runs out while waiting for call confirmation.
1732	Timer runs out while waiting for clear confirmation.
1733	Timer has run out while waiting a reset confirm.
1740	Call setup or call clearing problem.
1741	Open failure on virtual circuit. No flow control on this NS.
1742	Incorrect facility. Protocol error for the equipment directly connected to the Bull Datanet.
1744	Unknown subscriber.
1745	End of time-out on reset confirm. Invalid facility length. Protocol error for the equipment directly
1747	No logical channel available.
1749	End of time-out on call confirm.

174F	Incorrect packet length. Protocol error for the equipment directly connected to the Bull Datanet.
1755	Flow control, window, packet size or reset error.
1760	Frame disconnection.
1770	Frame connection.
1771	Frame reset.
1781	No more network routes available for X.25 switching.
1782	Maximum of 15 switches have been used,
1783	Flow control negotiation error.
1785	Frame level disconnection.
1786	Frame level connection.
1787	Frame level reset.
1790	Frame level not established.
1791	No more logical paths available for the PAD.
1792	Echo service busy.
1793	Incorrect PAD password.
1794	All the PAD virtual circuits are used
1795	X.25 initialization not possible.
179B	LCN not null in restart packet
179D	Incompatible header (receive error: all VC of concerned NS
179E	LCN greater than NBVC in NS directive
179F	Incorrect packet type
17A0	Invalid facility.
17B0	Normal disconnection.
17B1	X.25 Echo in use.
17B2	No more logical channels available.
17B3	No more PAD connections allowed.
17B4	TS SX25 or NU X25 object locked.
17B5	Buffer capacity overflow.
17B6	Normal disconnection.
17B8	Unknown calling SNPA (Sub-Network Point of Attachment).
17B9	Internet problem.
17CB	Call collision on VC
17CC	Incompatible generations (NR object without mapping).
17CE	Invalid status NR locked.
17CF	Lack of space.
17D0	Unknown subscriber.
17D4	TSCNX already used for another connection. SCF internal error.
17D7	Transport station locked.
17DD	Proper NS locked.
17DE	Invalid status NR locked.
17DF	Lack of space.

17E0	Forbidden parameter or invalid value.
17E1	Invalid transition.
17E2	Upward-mapped object (TS) not locked.
17E3	No object mapped above.
17E4	NR not locked (MP NR -ADD/-SUB) or virtual circuit already open.
17E5	NR is last in list and the TS is not locked.
17E6	No object mapped above (UP NR -PRIO). NR not mapped on TS.
17E7	Upward mapped object not locked
17E9	Mix of datagram and connection network
17EB	Class inconsistent with NR.
17EE	Incompatible generations. NR object without mapping.
17FF	Wrong parameter in administrative CALL
18xx	**Transport Layer**
1800	Normal disconnection initiated by the correspondent
1801	Local saturation at connection request time.
1802	Failed negotiation at connection time.
1803	Duplicate connection. Two or more requests have been issued for the same connection.
1804	Redundant request.
1805	Retransmission Time-out at transport level.
1806	Survey time-out at transport level.
1807	Transport protocol error.
1808	Session Control specified is not available (inaccessible).
1809	Requested Session Control Id unknown by remote transport.
180A	Termination because of disconnection by administration.
180B	Session Control/Transport interface error.
180C	Connection request on non-sharable VC in case of ISO Transport. ISO: header or parameter length is invalid.
1817	Station in shut-down state.
181F	No memory space at connection time.
1821	Session Control inaccessible by configured session routes. ISO: Session entity not attached to TSAP.
1824	Collision between Close NC and Open TC.
182E	Remote station not configured.
182F	Resource saturation.
1831	ISO: No route for the called NSAP.
1832	ISO: Received NSAP addresses are wrong.
1833	Segmentation violation.
1834	ISO:QOS priority not available temporarily, due to a local condition (for example, lack of resources).
1835	ISO:QOS priority permanently unavailable locally (for example, due to an error in the system generation).

183A	ISO: Remote reason not specified.
183C	ISO: Remote transport entity congestion at connect request time.
1840	Server in terminating state. TC has been re-assigned on another NC.
18A1	An additional NC has been assigned to a TC.
18B0	NC has been re-assigned on another VC.
18EF	Disconnection at Transport level caused by reception of RESTART DSA during the transfer phase.

Windows Sockets error Codes

Below is a list of Windows Sockets return codes and the corresponding description.

Hex code	Windows Sockets Access Error name	Description
2714	WSAEINTR	The (blocking) call was cancelled via WSACancelBlockingCall()
2719	WSAEBADF	The socket descriptor is not valid.
271E	WSAEFAULT	An invalid argument was supplied to the Windows Sockets API.
2726	WSAEINVAL	An invalid call was made to the Windows Sockets API.
2728	WSAEMFILE	No more file descriptors are available.
2733	WSAEWOULDBLOCK	The socket is marked as non-blocking and no connections are present to be accepted.
2734	WSAEINPROGRESS	A blocking Windows Sockets call is in progress.
2735	WSAEALREADY	The asynchronous routine being cancelled has already completed.
2736	WSAENOTSOCK	The descriptor is not a socket.
2737	WSAEDESTADDRREQ	A destination address is required.
2738	WSAEMSGSIZE	The datagram was too large to fit into the specified buffer and was truncated.
2739	WSAEPROTOTYPE	The specified protocol is the wrong type for this socket.
273A	WSAENOPROTOOPT	The option is unknown or unsupported.

273B	WSAEPROTONOSUPPORT	The specified protocol is not supported.
273C	WSAESOCKTNOSUPPORT	The specified socket type is not supported in this address family.
273D	WSAEOPNOTSUPP	The referenced socket is not a type that supports connection-oriented service.
273E	WSAEPFNOSUPPORT	
273F	WSAEAFNOSUPPORT	The specified address family is not supported by this protocol.
2740	WSAEADDRINUSE	The specified address is already in use.
2741	WSAEADDRNOTAVAIL	The specified address is not available from the local machine.
2742	WSAENETDOWN	The Windows Sockets implementation has detected that the network subsystem has failed.
2743	WSAENETUNREACH	The network address can't be reached from this host. There is probably a problem in the way you have set up TCP/IP routing for your PC (most likely you have not defined a default router).
2744	WSAENETRESET	The connection must be reset because the Windows Sockets implementation dropped it.
2745	WSAECONNABORTED	The connection has been closed.
2746	WSAECONNRESET	
2747	WSAENOBUFS	Not enough buffers available, or too many connections.
2748	WSAEISCONN	The socket is already connected.
2749	WSAENOTCONN	The socket is not connected.
274A	WSAESHUTDOWN	The socket has been shutdown.
274B	WSAETOOMANYREFS	
274C	WSAETIMEDOUT	Attempt to connect timed out without establishing a connection.
274D	WSAECONNREFUSED	The attempt to connect was forcefully rejected. The service on the other side is not available.
274E	WSAELOOP	Too many symbolic links were encountered in translating the path name.
274F	WSAENAMETOOLONG	

2750	WSAEHOSTDOWN	The host machine is out of service.
2751	WSAEHOSTUNREACH	The host machine is unreachable.
2752	WSAENOTEMPTY	
2753	WSAEPROCLIM	
2754	WSAEUSERS	
2755	WSAEDQUOT	
2756	WSAESTALE	
2757	WSAEREMOTE	
276B	WSASYSNOTREADY	Indicates that the underlying network subsystem is not ready for network communication.
276C	WSAVERNOTSUPPORTED	The version of Windows Sockets API support requested is not provided by this particular Windows Sockets implementation.
276D	WSANOTINITIALISED	A successful WSAStartup() must occur before using this API.
2AF9	WSAHOST_NOT_FOUND	Authoritative answer host not found.
2AFA	WSATRY_AGAIN	Non-authoritative answer host not found, or SERVERFAIL.
2AFB	WSANO_RECOVERY	Non-recoverable errors, FORMERR, REFUSED, NOTIMP.
2AFC	WSANO_DATA	Valid name, no data record of requested type.