PERICOM

*teem***talk** *& teem***X**

API Programmer's Guide

Pericom PLC

UK

The Priory, Cosgrove
Milton Keynes MK19 7JJ
Tel +44 (0) 1908 265533
Fax +44 (0) 1908 265534
EMAIL sales@pericom.co.uk
http://www.pericom-software.com

USA

Golden Crest Corporate Centre
2271 Highway 33 Suite 106
Hamilton Square NJ 08690
Tel +1 (609) 588 5300
Fax +1 (609) 588 8906
EMAIL sales@pericom-usa.com
http://www.pericom-software.com

FRANCE

1 rue James Joule
St Quentin-en-Yvelines
78286 Guyancourt Cedex
Tel +33 1 30 12 28 21
Fax +33 1 30 12 28 22
EMAIL sales@pericom-software.fr
http://www.pericom-software.com

GERMANY

Schlebusch Arcaden
Am Klösterchen 6
D-51375 Leverkusen
Tel +49 (0)214/85537-0
Fax +49 (0)214/85537-77
EMAIL sales@pericom.de
http://www.pericom-software.com

PERICOM

Solutions. Not answers.

Support Hotlines

United Kingdom:	**(01908) 265533**
Germany:	**0214 / 85537-0**
France:	**(0)1 30 12 28 21**
USA & Canada:	**1 (609) 588 5300**
Elsewhere:	**+44 1908 265533**

Dial one of the above numbers if you need help or advice on this software.

Software Version 3.10
September 1999

Part number: API-3.10

PERICOM PLC
The Priory, Cosgrove, Milton Keynes, MK19 7JJ
Tel: +44 (0) 1908 265533 Fax: +44 (0) 1908 265534

The material in this manual is for information purposes only and is subject to change without notice. Pericom PLC accepts no responsibility for any errors contained herein.

Trademarks

teemtalk and *teem*X are registered trademarks of Pericom PLC.
D410 is a trademark of Data General.
DEC, VT100 and VT320 are registered trademarks of Digital Equipment Corporation.
HP700/92 and HP2392A are trademarks of Hewlett Packard Company.
IBM is a registered trademark of International Business Machines Corporation.
Microsoft is a registered trademark, and Windows is a trademark of Microsoft Corporation.
Prime is a registered trademark and PT250 is a trademark of Prime Computer, Inc.
UNIX is a registered trademark of AT&T.
WYSE is a registered trademark of Wyse Technology Inc.
X Window System is a trademark of the Massachusetts Institute of Technology.
All other product names are trademarks of their respective manufacturers.

Contents

Notes

1

Introduction

This chapter introduces the Application Programming Interface (API) and describes the scope of this manual.

What Is API?

The Application Programming Interface (API) enables the *teemtalk* and *teem*X range of terminal emulators to be integrated into user written applications using a comprehensive set of commands. The API is compatible with the IBM HLLAPI interface, providing extensions to cater for all of the emulations supported by *teemtalk* and *teem*X.

The API interface consists of a C library which the user links with an application program. The application program can then connect to up to 26 instances of any suitable *teemtalk* or *teem*X emulator by invoking the API library. Each of these emulators executes on the current host computer and is free to connect to a target computer in exactly the same manner as a standard emulator.

The *teem* emulators for the API interface are similar to the standard *teem* emulators except that they have an additional communication channel to the API interface library. This interface channel is completely transparent to the user and the *teem* emulator can either be invoked from the command line as usual (with no connection to the API interface library), or from the *teem* API interface library (with the corresponding communication channel enabled). The user is free to specify any options that would normally be included on the command line for loading the *teem* emulator.

The Programmer's API Development Kit includes an executable file that enables you to perform any of the API functions. The name of the file is as follows:

teemtalk for Windows*:*	**HAPITEST.EXE**
teemtalk for Windows NT*:*	**APITST32.EXE**
*teem*X*:*	**APITEST**

The original source files are also supplied as API program examples. The contents of these files are shown in an appendix at the end of this manual.

About This Programmer's Manual

This manual consists of the following chapters and appendices:

Chapter 1: ***Introduction***
Introduces the Application Program Interface.

Chapter 2: ***Overview***
Provides an overview of the API interface and describes how to create and use an API program.

Chapter 3: ***API Functions***
Describes all the supported API functions in detail.

Appendix A: ***Field Attribute Representation***
Describes how field attributes are represented in the data string returned by a query field attribute function.

Appendix B: ***teemtalk API Program Examples***
Provides a hard copy of the original source files for the **HAPITEST.EXE** program, and shows example files used to generate an API link between *teemtalk* and a Visual Basic application.

Appendix C: ***teemX API Program Examples***
Provides a hard copy of the original source files for the **APITEST** program.

2

Overview

This chapter provides an overview of the API interface and describes how to create and use an API program.

The API Interface

The API can be used to simplify the creation of a local application which requires interaction with a Host system as a terminal. Typical uses might be to:

- Automate repetitive tasks, like automatic logon.
- Provide a “user-friendly” interface to a complex host application.
- Provide a single user interface to multiple host applications, on different host systems.
- Integrate existing host data into a new local application.

API for teemtalk for Windows 3.1

The *teemtalk* API is available via the **PCSHLL.DLL** file which is compatible with the DLL provided with IBM’s Personal Communications/3270 product. This DLL must be available in a directory in the standard Windows directory search path when *teemtalk* and/or your application is first started. During setup, it is copied into the same area as other *teemtalk* DLLs.

This DLL and any copy of *teemtalk* started with API support references an initialization file called **TTHLLAPI.INI**, which must also be available in a directory in the standard Windows directory search path. The creation of this file is described later. Each session/PSID (Presentation Space ID) which has a section in this file is available for use by your API program.

Note: Programs created using the 16-bit teemtalk for Windows version of API can be run by the 32-bit version of API for Windows 95 and NT.

If you want to start a session before you start up the API program, you must specify the following on the command line, where **A** is the session identifier :-

-hllapiA

This may be necessary for some API programs to run.

Creating Your Own API Program

In order to use the API in your own program, you must use a function call to the DLL for each command you want to issue. This takes the following form:

```
int FAR PASCAL hllapi(
                        LPINT    lpiFuncNum,
                        LPSTR    lpDataString,
                        LPINT    lpiDataStringLength,
                        LPINT    lpiRetCode
                       );
```

where:

lpiFuncNum is a pointer to an integer value containing the 'Function Number' parameter.

lpDataString is a pointer to the memory ('Data String' parameter) used to pass data to and from each HLLAPI Function.

lpiStringLength is a pointer to an integer value containing the 'Data String Length' parameter.

lpiRetCode is a pointer to an integer value containing the 'Presentation Space Position' parameter on entry to the function, and the 'Return Code' parameter on exit.

returns:

Return Code for the specific function, also in **lpiRetCode.**

The API is driven by function numbers, which identify the individual commands supported. The **hllapi()** function is prototyped in the **HAPIFUNC.H** file, which also contains definitions for all the Functions, Return Codes, etc. used by the API.

When you 'link' your application, you may include the **PCSHLL.LIB** library file to IMPORT the hllapi() function needed from the DLL, or else specify it explicitly in the ***.DEF** file in the IMPORT section as:

PCSHLL.hllapi

A sample application program called **HAPITEST.EXE** is provided which allows the user to interact with the API directly by issuing individual API functions. The source files for this program are also included and these are documented in the *teemtalk API Program Examples* appendix.

Creating An API Initialization File

In order to use a product that provides an API interface, you need to create an initialization file called **TTHLLAPI.INI** which must reside in a directory in the standard Windows directory search path. This can contain the following:

1. A **[Trace]** section, giving the file/path for debugging information.
2. A section for each API session, labelled **[A]** through **[Z]**.

The default **TTHLLAPI.INI** file in the **MISC** directory on the installation diskette contains the following:

[Trace]
Filename=

[A]
Program=c:\teemtalk\tt320iw.exe
SessionName=
Emulation=IBM3270,3278-2-E
Rows=24
Cols=80
CodePage=37

The optional **[Trace]** section defines the filename/path to be used for detailed debugging information on API execution.

Each session ID section (**[A] - [Z]**) specifies the default values for the session. The entries shown above are the defaults if an entry is missing.

The **Program=** line defines the program API will use to start the session if it is not already running.

SessionName= defines an override to the session name shown in the emulation window.

The **Emulation=** line defines the emulation to be used. If the IBM 3270 or IBM 5250 emulation is specified then the model number must follow the emulation name, with a comma separating the two. Valid emulation names are as follows (except for *teemtalk-1779W*):

ADDS A2	**IBM3270**	**TA6530**	**VT100**
Ansi BBS	**IBM5250**	**TVI910**	**VT300 7-Bit**
ATT4410	**ICL7561**	**TVI920**	**VT300 8-Bit**
BQ 3107	**MDI Prism-8**	**TVI925**	**WY50**
DG410	**MDI Prism-9**	**Viewdata 40**	**WY50+**
HP70092	**PT250**	**Viewdata 80**	**WY60**
HZ1500	**Siemens 97801**	**Viewdata Split**	
IBM3151	**Stratus V102**	**VT52**	

Valid IBM 3270 model numbers are as follows (except for *teemtalk-1779W*):

3278-2	**3278-5**	**3278-4-E**	**3279-3**	**3287-1**
3278-3	**3278-2-E**	**3278-5-E**	**3279-4**	
3278-4	**3278-3-E**	**3279-2**	**3279-5**	

Valid IBM 5250 model numbers are as follows (except for *teemtalk-1779W*):

5291-1	**3179-2**	**3477-FC**	**3487-HA**
5292-2	**3196-A1**	**3477-FG**	**3487-HC**
5251-11	**3180-2**	**3486-BA**	

Valid emulation names for *teemtalk-1779W* are as follows:

M1779DEC	**M1779ICL**	**M1779HPRATES**	**M1779PRS**
M1779IBM	**M1779HW**	**M1779CAPSS**	

The **Rows=** line defines the default number of screen rows for a non-active session. This is overridden when the session becomes active.

The **Cols=** line defines the default number of screen columns for a non-active session. This is overridden when the session becomes active.

The **CodePage=** line defines the default IBM code page (37 = US) for a non-active session. This is overridden when the session becomes active.

Initiating & Closing A Session

The following command will start a session without making a connection:

```
int    WINAPI    open_workstation(
                                    int       iShortname,
                                    LPSTR     lpArgs
                                    ) ;
```

where

iShortname is an integer value containing the Presentation Space shortname (**A** to **Z**) required.

lpArgs is a pointer to any extra arguments to be used when launching *teemtalk*.

returns:	**≥0**	OK (= process ID)	**-8**	Session already exists
	-1	Invalid presentation space ID was specified	**-9**	System error
			-11	Resource unavailable

This command sets the session parameter for USENEW, does a connect, then sets the session parameter for USEOLD and disconnects. (See the description of **Set Session Parameters** (9) for further details).

To close a session, issue the following command:

```
int    WINAPI    close_workstation(
                                    int    iShortname
                                    ) ;
```

where **iShortname** is an integer value containing the Presentation Space shortname (**A** to **Z**) required.

returns:	**≥0**	OK (= process ID)	**-9**	System error
	-1	Invalid presentation space ID was specified	**-11**	Resource unavailable

This command sets the session parameter for USEOLD, does a connect, then sets the session parameter for USENEW and disconnects.

Both commands are prototyped in the **HAPIFUNC.H** file.

API for teemtalk for Windows 95 & NT

The *teemtalk* API is available via the **PCSHLL32.DLL** file which is compatible with the DLL provided with IBM's Personal Communications/3270 product. This DLL must be available in a directory in the standard Windows directory search path when *teemtalk* and/or your application is first started. During installation, one of the following files is copied into the same area as other *teemtalk* DLLs and renamed **PCSHLL32.DLL**:

PCSHLL95.DLL	for Windows 95
PCSHLLNT.DLL	for Windows NT

Note: Programs created using the 16-bit teemtalk for Windows version of API can be run by the 32-bit version of API for Windows 95 and NT.

If you want to start a session before you start up the API program, you must specify the following on the command line, where **A** is the session identifier :-

-hllapiA

This may be necessary for some API programs to run.

Creating Your Own API Program

In order to use the API in your own program, you must use a function call to the DLL for each command you want to issue. This takes the following form:

```
__declspec(dllexport)
WORD  WINAPI  hllapi(
                        LPWORD      lpwFuncNum,
                        LPBYTE      lpDataString,
                        LPWORD      lpwDataStringLength,
                        LPWORD      lpwRetCode
                       );
```

where:

lpwFuncNum is a pointer to a WORD (16-bit unsigned) value containing the 'Function Number' parameter.

lpDataString is a pointer to the memory ('Data String' parameter) used to pass data to and from each HLLAPI function.

lpwStringLength is a pointer to a WORD (16-bit unsigned) value containing the 'Data String Length' parameter.

lpwRetCode is a pointer to to a WORD (16-bit unsigned) value containing the 'Presentation Space Position' parameter on entry to the function, and the 'Return Code' parameter on exit.

returns:

Return Code for the specific function, also in **lpwRetCode.**

The API is driven by function numbers, which identify the individual commands supported. The **hllapi()** function is prototyped in the **HAPIFUNC.H** file, which also contains definitions for all the Functions, Return Codes, etc. used by the API.

When you 'link' your application, you may include the **PCSHLL32.LIB** library file to IMPORT the **hllapi()** function needed from the DLL, or else specify it explicitly in the ***.DEF** file in the IMPORT section as:

PCSHLL32.hllapi

A sample application program called **APITST32.EXE** is provided which allows the user to interact with the API directly by issuing individual API functions. The source files for this program are also included and these are documented in the *teemtalk API Program Examples* appendix.

Registry Entries

In order to use a product that provides an API interface, *teemtalk* and the **PCSHLL32.DLL** use the environment variable **TTAPICFG** to access the Registry section for the user. For example, for Windows NT, the section created by default is **Software\Pericom\TTHLLAPI** under **HKEY_LOCAL_MACHINE**. This section can contain a section for each API session, **A** through **Z**. It contains the following by default:

Variable	Type	Value
Program	**REG_SZ**	**"c:\program files\teemtalk\tt320iw.exe"**
SessionName	**REG_SZ**	**""**
Emulation	**REG_SZ**	**"IBM3270,3278-2-E"**
Rows	**REG_DWORD**	**24**
Cols	**REG_DWORD**	**80**
CodePage	**REG_DWORD**	**37**

If you want to start a session before you start up the API program, you must specify **-hllapiA** on the command line, where **A** is the session ID. This may be necessary for some API programs to run. These entries define the available session IDs for API programs to use. If a session is not already running when required, the information for that session ID will be taken from this file. There MUST be an entry for each requested session, which MUST contain at least the **Program** entry.

Each session ID section provides the default values for the session. The entries shown earlier are the defaults if an entry is missing.

The **Program** line defines the program API will use to start the session if it is not already running.

The **SessionName** line defines an override to the session name shown in the emulation window.

The **Emulation** line defines the emulation to be used. If the IBM 3270 or IBM 5250 emulation is specified then the model number must follow the emulation name, with a comma separating the two. Valid emulation names are as follows (except for *teemtalk-1779W*):

ADDS A2	**IBM3270**	**TA6530**	**VT100**
Ansi BBS	**IBM5250**	**TVI910**	**VT300 7-Bit**
ATT4410	**ICL7561**	**TVI920**	**VT300 8-Bit**
BQ 3107	**MDI Prism-8**	**TVI925**	**WY50**
DG410	**MDI Prism-9**	**Viewdata 40**	**WY50+**
HP70092	**PT250**	**Viewdata 80**	**WY60**
HZ1500	**Siemens 97801**	**Viewdata Split**	
IBM3151	**Stratus V102**	**VT52**	

Valid IBM 3270 model numbers are as follows (except for *teemtalk-1779W*):

3278-2	**3278-5**	**3278-4-E**	**3279-3**	**3287-1**
3278-3	**3278-2-E**	**3278-5-E**	**3279-4**	
3278-4	**3278-3-E**	**3279-2**	**3279-5**	

Valid IBM 5250 model numbers are as follows (except for *teemtalk-1779W*):

5291-1	**3179-2**	**3477-FC**	**3487-HA**
5292-2	**3196-A1**	**3477-FG**	**3487-HC**
5251-11	**3180-2**	**3486-BA**	

Valid emulation names for *teemtalk-1779W* are as follows:

M1779DEC	**M1779ICL**	**M1779HPRATES**	**M1779PRS**
M1779IBM	**M1779HW**	**M1779CAPSS**	

The **Rows** line defines the default number of screen rows for a non-active session. This is overridden when the session becomes active.

The **Cols** line defines the default number of screen columns for a non-active session. This is overridden when the session becomes active.

The **CodePage** line defines the default IBM code page (37 = US) for a non-active session. This is overridden when the session becomes active.

When a *teemtalk* for Windows NT session is started, the following entries are added to the Registry section under the API session section:

Variable	Type	Value	
ProcessID	**REG_DWORD**	***1234***	(current process)
WindowID	**REG_DWORD**	***5678***	(main *teemtalk* window)
WindowTitle	**REG_SZ**	***Title***	

These entries will be deleted when the session is closed.

When your program connects to a session, the **PCSHLL32.DLL** will add the following entry:

Pipe **REG_SZ** ***pipename***

This entry will be deleted when the session is disconnected. It will also add the entry:

MakeConnection **REG_SZ** **Yes**

which is deleted by *teemtalk* when it recognizes it.

Initiating & Closing A Session

The following command will start a session without making a connection:

```
int    WINAPI    open_workstation(
                                    int       iShortname,
                                    LPSTR     lpArgs
                                    ) ;
```

where

iShortname is an integer value containing the Presentation Space shortname (**A** to **Z**) required.

lpArgs is a pointer to any extra arguments to be used when launching *teemtalk*.

returns:	**≥0**	OK (= process ID)	**-8**	Session already exists
	-1	Invalid presentation space ID was specified	**-9**	System error
			-11	Resource unavailable

This command sets the session parameter for USENEW, does a connect, then sets the session parameter for USEOLD and disconnects. (See the description of **Set Session Parameters** (9) for further details).

To close a session, issue the following command:

```
int    WINAPI    close_workstation(
                                    int   iShortname
                                    ) ;
```

where **iShortname** is an integer value containing the Presentation Space shortname (**A** to **Z**) required.

returns:	**≥0**	OK (= process ID)	**-9**	System error
	-1	Invalid presentation space ID was specified	**-11**	Resource unavailable

This command sets the session parameter for USEOLD, does a connect, then sets the session parameter for USENEW and disconnects. (See the description of **Set Session Parameters** (9) for further details).

Both commands are prototyped in the **HAPIFUNC.H** file.

*API for teem*X

The *teem*X API is available by compiling your own application using the supplied **hapilib.a** library, which is mainly compatible with other IBM HLLAPI products, but extended for use with the other alpha emulations supported by *teem*X.

Copies of *teem*X started with API support can reference a configuration file pointed to by the **ENV**ironment variable **TXAPICFG**. The creation of this file is described later. This is used to determine which API sessions are defined and to place information for the API library and the *teem*X API sessions to initiate linkage. This can connect to *teem*X sessions that are already running, and does not destroy the sessions when your API program completes.

If you want to start a session before you start up the API program, you must specify the following on the command line, where **A** is the session identifier:

-hllapisession A

This may be necessary for some API programs to run.

Alternatively, when a configuration file is not defined by the **ENV**ironment variable **TXAPICFG**, there is a different method for using the API. With this method each

application program can connect to up to 26 sessions/copies of a *teem*X emulator, which are specified on each initial **Connect Presentation Space** (1) function call for each session/PSID. These are shut down/killed individually by the **Disconnect Presentation Space** (2) function call, or all connected sessions are disconnected with the **Reset System** (21) function call. There is no need to disconnect from all instances of *teem*X before exiting, but they will remain functional after the HLLAPI application terminates if not.

Creating Your Own API Program

In order to use the API in your own program, you will use a function call to the *teem*X HLLAPI library for each command you wish to issue. This takes the following form:

```
int  hllc(
     int     *lpiFuncNum,
     char    *lpDataString,
     int     *lpiDataStringLength,
     int     *lpiRetCode
     );
```

where:

lpiFuncNum is a pointer to an integer value containing the 'Function Number' parameter.

lpDataString is a pointer to the memory ('Data String' parameter) used to pass data to and from each HLLAPI Function.

lpiStringLength is a pointer to an integer value containing the 'Data String Length' parameter.

lpiRetCode is a pointer to an integer value containing the 'Presentation Space Position' parameter on entry to the Function, and the 'Return Code' parameter on exit.

returns:

Return Code for specific Function, also in **lpiRetCode.**

The API is driven by these 'Function Numbers', which identify the individual commands supported. The **hllc()** function is prototyped in the **HAPIFUNC.H** file, which also contains definitions for all the Functions, Return Codes, etc. used by the API.

Your application can be compiled using the command

cc -test test.c hapilib.a

or any more complex compile that your compiler allows.

A sample application program called **APITEST** is provided which allows the user to interact with the API directly by issuing individual API functions. The source files for this program are also included and these are documented in the *teem*X *API Program Examples* appendix.

Creating An API Configuration File

An API configuration file is used to determine which API sessions are defined and to place interface information for the API library and *teem*X API sessions to initiate linkage. The **ENV**ironment variable **TXAPICFG** is used to point to the API configuration file. Any *teem*X API sessions started before your API program will require the **ENV**ironment variable, or you can specify/override the name of the configuration file on the command line using:

-hllapiconfig filename

The entries in the API configuration file define the session IDs that are available for API programs to use. If a session is not already running when required, the information for that session ID will be taken from this file. There MUST be an entry for each requested session, which MUST contain at least the **Program=** entry.

A typical API configuration file contains the following:

[A]
Program=./xt320
SessionName=Name
Emulation=IBM3270,3278-2-E
Rows=24
Cols=80
CodePage=37

Each session ID section (**[A] - [Z]**) specifies the default values for the session. The entries shown above are the defaults if an entry is missing.

The **Program=** line defines the program API will use to start the session if it is not already running.

SessionName= defines an override to the session name shown in the emulation window.

The **Emulation=** line defines the emulation to be used. If the IBM 3270 or IBM 5250 emulation is specified then the model number must follow the emulation name, with a comma separating the two. Valid emulation names are as follows (except for *teem*X-*1779*):

ADDS A2	**IBM3270**	**TA6530**	**VT100**
Ansi BBS	**IBM5250**	**TVI910**	**VT300 7-Bit**
ATT4410	**ICL7561**	**TVI920**	**VT300 8-Bit**
BQ 3107	**MDI Prism-8**	**TVI925**	**WY50**
DG410	**MDI Prism-9**	**Viewdata 40**	**WY50+**
HP70092	**PT250**	**Viewdata 80**	**WY60**
HZ1500	**Siemens 97801**	**Viewdata Split**	
IBM3151	**Stratus V102**	**VT52**	

Valid IBM 3270 model numbers are as follows (except for *teem*X-*1779*):

3278-2	**3278-5**	**3278-4-E**	**3279-3**	**3287-1**
3278-3	**3278-2-E**	**3278-5-E**	**3279-4**	
3278-4	**3278-3-E**	**3279-2**	**3279-5**	

Valid IBM 5250 model numbers are as follows (except for *teem*X-*1779*):

5291-1	**3179-2**	**3477-FC**	**3487-HA**
5292-2	**3196-A1**	**3477-FG**	**3487-HC**
5251-11	**3180-2**	**3486-BA**	

Valid emulation names for *teem*X-*1779* are as follows:

M1779DEC	**M1779ICL**	**M1779HPRATES**	**M1779PRS**
M1779IBM	**M1779HW**	**M1779CAPSS**	

The **Rows=** line defines the default number of screen rows for a non-active session. This is overridden when the session becomes active.

The **Cols=** line defines the default number of screen columns for a non-active session. This is overridden when the session becomes active.

The **CodePage=** line defines the default IBM code page (37 = US) for a non-active session. This is overridden when the session becomes active.

When a *teem*X API session is started, the following entries will be added to the configuration file:

ProcessID=1234
WindowID=5678
DisplayID=9012
WindowTitle=Title

These entries will be deleted when the session is closed.

The API library used by your program will add the entries:

hllchild_to_parent=/tmp/config.Acp
hllparent_to_child=/tmp/config.Apc

when it is connected to a session, and delete them when it is disconnected. It will also add the entry:

MakeConnection=Yes

which is deleted by the *teem*X program once it has recognized it.

Initiating & Closing A Session

The following command will start a session without making a connection:

```
int         open_workstation(
                        int         iShortname,
                        char        *lpArgs
                        ) ;
```

where

iShortname is an integer value containing the Presentation Space shortname (**A** to **Z**) required.

lpArgs is a pointer to any extra arguments to be used when launching *teem*X.

This command sets the session parameter for USENEW, does a connect, then sets the session parameter for USEOLD and disconnects. (See the description of **Set Session Parameters** (9) for further details).

To close a session, issue the following command:

```
int         close_workstation(
                        int     iShortname
                        ) ;
```

where **iShortname** is an integer value containing the Presentation Space shortname (**A** to **Z**) required.

This command sets the session parameter for USEOLD, does a connect, then sets the session parameter for USENEW and disconnects. (See the description of **Set Session Parameters** (9) for further details).

Both commands are prototyped in the **HAPIFUNC.H** file.

The Presentation Space

A presentation space (PS) is the portion of computer memory that is used to display information in the terminal emulation window. Each instance of *teemtalk* or *teem*X (i.e. each session) has its own presentation space. The API supports up to 26 presentation spaces and interacts with these one at a time.

Identification

Each presentation space is identified by a unique name (a PSID) which is a single character in the range **A** through **Z**. Some functions require the PSID to be specified in a preceding call to **Connect Presentation Space** (1), while others include the PSID as part of their calling data string.

Making A Connection

The **Connect Presentation Space** (1) function is used to make an initial connection between your API program and an instance of *teemtalk* or *teem*X, using a PSID to identify it. Future re-connects to an instance of the terminal emulator just require the PSID to be specified. The currently connected presentation space can be disconnected by the **Disconnect Presentation Space** (2) function, or you can disconnect all the presentation spaces launched by the API application by issuing **Reset System** (21).

Addressing

Some functions require a specific location within the presentation space to be specified in their calling data string. Presentation space addressing relates to the screen size of the current session. A PS position is specified as the offset into the presentation space working left to right, line by line from top to bottom, where position 1 is the top left corner of the screen display (row 1 and column 1) and the highest position number is the bottom right corner (e.g. 1920 for a display size of 24 rows by 80 columns). The size of the currently connected presentation space can be determined by a **Query Session Status** (22) call and examining bytes 12 to 15 of the returned data string.

ASCII Mnemonics

ASCII mnemonics are used by the **Send Keystring** (3) and **Get Key** (51) functions to represent keyboard functions or codes that may not be represented by ASCII values or an available key. For example, the keyboard for one session might not be capable of generating key codes that are required by another session, so ASCII mnemonics representing these codes would be included in the data string sent by the **Send Keystring** (3) function.

All defined keys are represented by either a 1-byte ASCII value from the 256-character ASCII character set (e.g. **a** for the **a** key), or a 2-, 4-, or 6-byte ASCII mnemonic for function keys (e.g. **@r** for the **Ctrl** key, **@A@T** for **Print Screen**).

The escape character is represented by the ASCII character **@** (at) by default. You can change this to any other displayable ASCII character by using the **ESC=** option of the **Set Session Parameters** (9) function. The tables of mnemonics shown below use **@** to represent the escape character.

Mnemonics for unshifted keys and the shift keys **Upper Shift**, **Alt** and **Ctrl** consist of the escape character followed by an abbreviation. Mnemonics for shifted keys consist of the mnemonic for the shift indicator followed by the mnemonic for the unshifted key. Shift indicators are represented by the following 2-byte ASCII mnemonics:

Upper Shift:	@S
Alt:	@A
Ctrl:	@r

These shift indicator mnemonics are always accompanied by a non-shift-indicator character or mnemonic, they are never sent or received by an application on their own.

The following tables show the mnemonics for keys defined with functions. Please note that the mnemonics are case sensitive. For example, **@N** would refer to the **New Line** key function, whereas **@n** would refer to the **PF23** key function. AID key mnemonics are indicated by an asterisk (*).

Standard IBM Keyboard Mapping

The following ASCII mnemonics are valid by default or when the **KEYSIBM** option is specified using the **Set Session Parameters** (9) function.

@B	Backtab	@L	Cursor Left	@Y	Caps Lock
@C	Clear *	@N	New Line	@Z	Cursor Right
@D	Delete	@P	Print	@s	Scroll Lock
@E	Enter *	@R	Reset	@t	Num Lock
@F	Erase EOF	@T	Tab	@@	@ (at) symbol
@I	Insert	@U	Cursor Up	@$	Alternate Cursor
@J	Jump	@V	Cursor Down	@<	Backspace Erase

*Note: The mnemonics for **Caps Lock**, **Scroll Lock** and **Num Lock** can be intercepted by the function **Get Key** (51), but they cannot be sent by **Send Keystring** (3).*

@0	Home	@7	PF7 *	@e	PF14 *	@l	PF21 *
@1	PF1 *	@8	PF8 *	@f	PF15 *	@m	PF22 *
@2	PF2 *	@9	PF9 *	@g	PF16 *	@n	PF23 *
@3	PF3 *	@a	PF10 *	@h	PF17 *	@o	PF24 *
@4	PF4 *	@b	PF11 *	@i	PF18 *	@x	PA1 *
@5	PF5 *	@c	PF12 *	@j	PF19 *	@y	PA2 *
@6	PF6 *	@d	PF13 *	@k	PF20 *	@z	PA3 *

@A@F	Erase Input	@A@b	Underscore
@A@H	System Request *	@A@y	Forward Word Tab
@A@J	Cursor Select *	@A@z	Backward Word Tab
@A@Q	Attention *	@S@x	Dup
@A@R	Device Cancel	@S@y	Field Mark
@A@T	Print Screen		

*Note: The mnemonic for the at symbol (@@) consists of the escape character followed by the literal @ symbol. If the escape character is represented by **%**, then the mnemonic would change to **%@**. The **ESC=** option of the **Set Session Parameters** (9) function is used to specify the character that represents escape.*

If you send either of the Print Screen mnemonics, place it at the end of the calling data string.

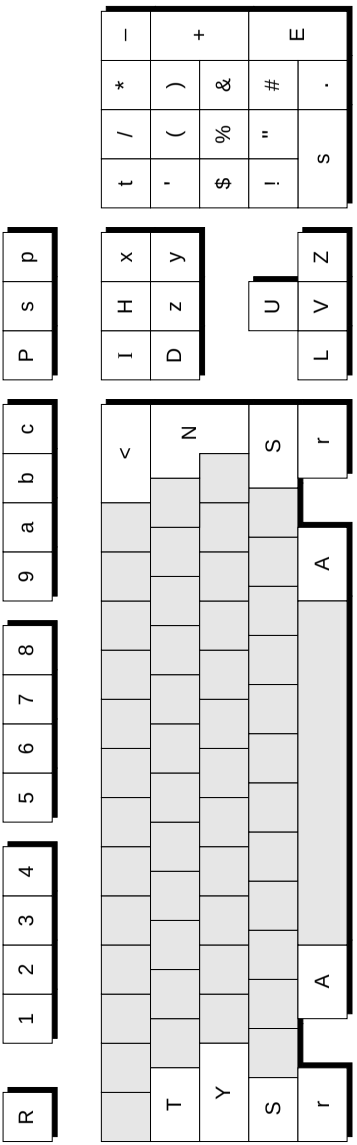
If you send the Device Cancel mnemonic, mnemonics are passed through with no error message. Local copy will not be affected.

Direct Keyboard Mapping

The following ASCII mnemonics are valid when the **KEYSKBD** option is specified using the **Set Session Parameters** (9) function.

@sp	Keypad 0	@@	@	@`	
@!	Keypad 1	@A	Alt	@a	F10
@"	Keypad 2	@B	Attn (122)	@b	F11
@#	Keypad 3	@C	Compose (LK250)	@c	F12
@$	Keypad 4	@D	Delete	@d	F13 (122)
@%	Keypad 5	@E	Enter	@e	F14 (122)
@&	Keypad 6	@F	Clear (122)	@f	F15 (122)
@'	Keypad 7	@G	Cursor Sel (122)	@g	F16 (122)
@(	Keypad 8	@H	Home	@h	F17 (122)
@)	Keypad 9	@I	Insert	@i	F18 (122)
@*	Keypad *	@J	ErInp (122)	@j	F19 (122)
@+	Keypad +	@K	ExSel (122)	@k	F20 (122)
@,	Keypad , (LK250/122)	@L	Cursor Left	@l	F21 (122)
@-	Keypad -	@M	Erase EOF (122)	@m	F22 (122)
@.	Keypad .	@N	Newline	@n	F23 (122)
@/	Keypad /	@O	Print (122)	@o	F24 (122)
@0		@P	Print Screen	@p	Pause
@1	F1	@Q	Play (122)	@q	Reset (122)
@2	F2	@R	Escape	@r	Control
@3	F3	@S	Shift	@s	Scroll Lock
@4	F4	@T	Tab	@t	Num Lock
@5	F5	@U	Cursor Up	@u	Enter (122)
@6	F6	@V	Cursor Down	@v	
@7	F7	@W	Zoom (122)	@w	
@8	F8	@X	Noname (122)	@x	Page Up
@9	F9	@Y	Caps Lock	@y	Page Down
@:	Hold (LK250)	@Z	Cursor Right	@z	End
@;	Print (LK250)	@[	PA1 (122)	@{	
@<	Backspace	@\	Backtab (122)	@|	
@=	Setup (LK250)	@]	PA2 (122)	@}	
@>	Datatalk (LK250)	@^	PA3 (122)	@~	
@?	Break (LK250)	@_	Home (122)		

Enhanced AT Keyboard Layout

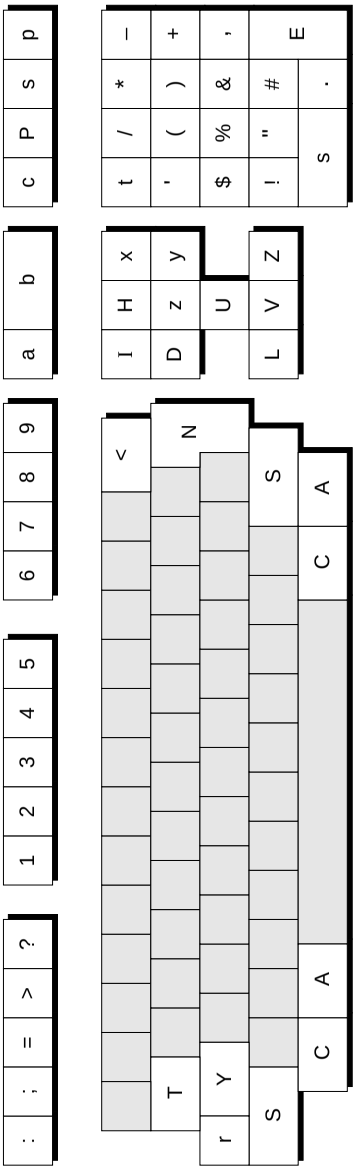


These ASCII mnemonics are valid when the **KEYSKBD** option is specified using the **Set Session Parameters** (9) function. The characters must be preceded by the escape character (@ by default).

122-Key Keyboard Layout

These ASCII mnemonics are valid when the **KEYSKBD** option is specified using the **Set Session Parameters** (9) function. The characters must be preceded by the escape character (@ by default).

LK250 Keyboard Layout



These ASCII mnemonics are valid when the **KEYSKBD** option is specified using the **Set Session Parameters** (9) function. The characters must be preceded by the escape character (@ by default).

teemtalk-1779W & teemX-1779

The following tables list the ASCII mnemonics for *teemtalk-1779W* and *teem*X*-1779*. AID key mnemonics are indicated by an asterisk (*).

	Keypad 0	@@	@	@`	Alt Gr
@!	Keypad 1	@A	Alt	@a	PF10 *
@"	Keypad 2	@B	Backtab *	@b	PF11 *
@#	Keypad 3	@C	Clear *	@c	PF12 *
@$	Keypad 4	@D	Delete *	@d	PF13 *
@%	Keypad 5	@E	Enter *	@e	PF14 *
@&	Keypad 6	@F	Erase EOF *	@f	PF15 *
@'	Keypad 7	@G		@g	PF16 *
@(	Keypad 8	@H		@h	PF17 *
@)	Keypad 9	@I	Insert *	@i	PF18 *
@*		@J		@j	PF19 *
@+		@K		@k	PF20 *
@,	Keypad Tab	@L	Cursor Left *	@l	PF21 *
@-	Keypad -	@M	Return *	@m	PF22 *
@.	Keypad .	@N	Newline *	@n	PF23 *
@/		@O		@o	PF24 *
@0	Home *	@P	Print *	@p	
@1	PF1 *	@Q	Attention *	@q	
@2	PF2 *	@R	Reset *	@r	Control
@3	PF3 *	@S	Shift	@s	
@4	PF4 *	@T	Tab *	@t	F1 *
@5	PF5 *	@U	Cursor Up *	@u	F2 *
@6	PF6 *	@V	Cursor Down *	@v	F3 *
@7	PF7 *	@W		@w	F4 *
@8	PF8 *	@X		@x	PA1 *
@9	PF9 *	@Y		@y	PA2 *
@:	Cursor Select *	@Z	Cursor Right *	@z	Delete Line
@;	Ex Sel *	@[	Escape *	@{	
@<	Backspace Erase *	@\	Backspace *	@|	
@=		@]		@}	
@>		@^		@~	
@?	Break *				

Special combinations:

@`@1	Blank Key 1 *	@r@P	Print Indent *
@`@2	Blank Key 2 *	@r@Q	System Request *
@`@3	Blank Key 3 *	@r@R	Device Cancel *
@`@4	Blank Key 4 *	@r@U	Shift Lock
@`@5	Blank Key 5 *	@r@\	Test *
@`@6	Blank Key 6 *	@S@x	Dup *
@`@7	Blank Key 7 *	@S@y	Field Mark *
@`@8	Blank Key 8 *	@S@[	Erase Input *

3

API Functions

This chapter describes all the API functions in detail.

Conventions

An API function is referred to by its function name which is followed by the function number in parentheses. Function descriptions are arranged alphabetically by name.

Each function description includes the following sections if applicable:

Prerequisite Calls

This section specifies the function(s) that must be called before the function described can take effect. If no prerequisite calls are required then *None* will be shown.

Supplied Parameters

This section lists and describes the parameters that must be defined for the function. If a parameter is not used then *NA* (not applicable) will be shown.

Returned Parameters

This section lists and explains the parameters that your program must receive following the function call.

Remarks

This section supplies technical information on the function and specifies any session options that may affect it.

Emulation Support Of Functions

The API functions described in this chapter can be used in all the terminal emulations supported by *teemtalk* and *teem*X with the following exceptions.

The functions listed below are not supported by VT52, ANSI-BBS, DG410, AT&T4410, Stratus V102, or the Viewdata modes:

Query Field Attribute (14)
Search Field (30)
Find Field Position (31)
Find Field Length (32)
Copy String To Field (33)
Copy Field To String (34)
Query Formatted Field Attribute (114)

These 'field' functions will only work in the emulations listed in the table below if the presentation space is formatted as specified:

Emulation	Considered Formatted When
VT100 **VT320 (7-bit)** **VT320 (8-bit)**	In Edit mode, plus ‘host characters protected’ and ‘NOT erase all’ set.
HP700/92	Format mode ON.
PT250	In Block mode, plus ‘logical attributes’ mode ON.
TA6530	In Block Protect mode, and a page is displayed.
IBM 3270	Presentation Space formatted.
IBM 5250	Connected.
IBM 3151	Presentation Space formatted.
Wyse Modes	In Block mode with Protect ON.
ICL 7561	Presentation Space formatted.

Function Descriptions

The API functions described on the following pages in alphabetical name order are listed below according to function number:

1 Connect Presentation Space
2 Disconnect Presentation Space
3 Send Keystring
4 Wait
5 Copy Presentation Space
6 Search Presentation Space
7 Query Cursor Location
8 Copy Presentation Space To String
9 Set Session Parameters
10 Query Sessions
11 Reserve
12 Release
13 Copy OIA
14 Query Field Attribute
15 Copy String To Presentation Space
18 Pause
20 Query System
21 Reset System
22 Query Session Status
23 Start Host Notification
24 Query Host Update
25 Stop Host Notification
30 Search Field
31 Find Field Position
32 Find Field Length
33 Copy String To Field
34 Copy Field To String
40 Set Cursor
50 Start Keystroke Intercept
51 Get Key
52 Post Intercept Status
53 Stop Keystroke Intercept
99 Convert Position / RowCol
103 Send Keypress
105 Copy Formatted Presentation Space
108 Copy Formatted Presentation Space To String
109 Set *teemtalk*/*teem*X Parameter
110 Get *teemtalk*/*teem*X Parameter
114 Query Formatted Field Attribute

CONNECT PRESENTATION SPACE (1)

teemtalk

This function establishes a connection between any API application program and the presentation space (PS), excluding it from use by any other API application program.

*teem*X

If an API configuration file is used, this function establishes a connection between any API application program and the presentation space (PS), excluding it from use by any other API application program.

If an API configuration file is not used, this function establishes a connection with the presentation space (PS) of a new or previously active *teem*X session. The initial Connect function launches an instance of *teem*X (which can include command line options) and gives it a unique identifying character. This character (referred to as the PSID) is used to identify that particular presentation space for future reconnection.

Prerequisite Calls

None.

Supplied Parameters

Function Number: Must be 1.

Data String: ***teemtalk:*** For the initial connection to a presentation space, this is a one character identifier (PSID) in the range **A** through **Z** (upper or lowercase).

***teem*X:** As above, but if an API configuration file is not used then the PSID must be followed by the executable name of the *teem*X product to be launched and any valid command line options.

***teemtalk & teem*X:** For reconnecting to an existing presentation space, this is the one character identifier (PSID) which identifies the particular session as specified by the initial connection.

Data String Length: ***teemtalk:*** NA. (1 is implied.)

***teem*X:** The length of the Data String in bytes. This parameter is not applicable if in end of transmission (EOT) mode.

PS Position: NA.

Returned Parameters

Return Code: The following codes are valid:

Code	Explanation
0	The function was successful; the presentation space is connected and unlocked.
1	An invalid presentation space ID was specified.
4	Successful, but host input available.
5	Successful, but host PS locked.
8	Either USENEW is set and there is an existing session, or USEOLD is set and there isn't an existing session. (See the **Set Session Parameters** (9) function.)
9	A system error occurred.
11	Resource unavailable. The PSID is already being used by another API application.

Remarks

teemtalk

An API application program can connect to any or all of the available PSIDs defined in the **TTHLLAPI.INI** file or Windows NT Registry, as long as no other API application program is using it. If a valid PSID is not already running as a *teemtalk* session, it will also be launched by this command, using the parameters in the **TTHLLAPI.INI** file or Windows NT Registry for this session. The currently connected PSID can be disconnected by a **Disconnect Presentation Presentation Space** (2).

*teem*X

An API application program can connect to any or all of the available PSIDs defined in the API configuration file, as long as no other API application program is using it. If a valid PSID is not already running as a *teem*X session, it will also be launched by this command, using the parameters in the API configuration file for this session. The currently connected PSID can be disconnected by a **Disconnect Presentation Presentation Space** (2).

If an API configuration file is not used then the API application can launch up to 26 instances of *teem*X. Most API commands are directed towards the most recently connected (or reconnected) instance of *teem*X (see specific commands for details). Instances of *teem*X launched by another API application or by the user cannot be accessed by the current API application. The currently connected instance of *teem*X

launched by the API application can be terminated by a **Disconnect Presentation Space** (2). All instances of *teem*X launched by the API application will be terminated by a **Reset System** (21).

If the user terminates an instance of *teem*X launched by the API application, the next API command issued to that presentation space will result in a return code of **9** (system error).

CONVERT POSITION / ROWCOL (99)

This function converts the host presentation space positional value into the display row/column coordinates, or vice versa. Note that the cursor position remains unchanged.

Prerequisite Calls

Connect Presentation Space (1)

Supplied Parameters

Function Number: Must be 99.

Data String: The presentation space ID character plus **P** for convert position to row/column coordinates, or **R** for convert row/column coordinates to presentation space position.

Data String Length: Not applicable when **P** specified in the data string. When **R** is specified, this is the row number between 1 and maximum.

PS Position: For **P**, this is the positional value between 1 and maximum. For **R**, this is the column number between 1 and maximum.

Returned Parameters

Data String Length: For **P**, the row number between 1 and maximum. For **R**, **0** if specified row number was invalid.

Return Code: The following codes are valid:

Code	Explanation
0	Specified position/row/column number invalid.
>0	This is the position or column.
9998	An invalid host presentation space ID was specified, or a system error occurred.
9999	Second character in the data string is neither **P** nor **R**.

Remarks

To determine how many rows and columns there are in the currently connected presentation space, use the **Query Session Status** (22) function.

COPY FIELD TO STRING (34)

This copies the contents of a specified field in the presentation space to a string. The function only applies to field formatted presentation spaces and can be used with both protected and unprotected fields. The location and length of a field can be found by using the **Find Field Position** (31) and **Find Field Length** (32) functions.

Prerequisite Calls

Connect Presentation Space (1)

Supplied Parameters

Function Number: Must be 34.

Data String: Predefined buffer to hold string (twice the length requested if the EAB option has been specified).

Data String Length: The number of characters to be copied.

PS Position: Any character position within the source field in the presentation space. (Note that the copy will always start at the beginning of the field.)

Returned Parameters

Data String: Characters from the source string in the presentation space. The function will copy up to the data string length specified or the end of the source field, whichever occurs first.

Data String Length: The number of characters actually copied.

Return Code: The following codes are valid:

Code	Explanation
0	The function was successful.
1	Not currently connected to a presentation space.
2	Data string length 0 specified.
6	Incorrect field size specified. The returned data string may be truncated.
7	Invalid presentation space position.
9	A system error occurred.
24	The presentation space is unformatted.

Remarks

This function is not supported by VT52, ANSI-BBS, DG410, AT&T 4410, Stratus V102, or the Viewdata modes. It is only valid when the presentation space is field formatted. Field formatting is emulation dependent as shown in the table on page 3-2.

Characters copied from the field are translated into 8-bit ASCII characters (ISO Latin-1) for the returned data string. This function is affected by the EAB/NOEAB and XLATE/NOXLATE session options. See **Set Session Parameters** (9).

The IBM 5250 emulation supports a presentation space of 24 rows by 80 columns. When an error message is received from the host or when the operator presses the **SysReq** key, a 25th row is displayed. When row 25 is displayed, it is a valid area for this function.

COPY FORMATTED PRESENTATION SPACE (105)

Copies the internal 4-byte representation of the currently connected formatted presentation space to a string.

Prerequisite Calls

Connect Presentation Space (1)

Supplied Parameters

Function Number: Must be 105.

Data String: Predefined buffer, at least four times the size of the presentation space.

Data String Length: Length of data string.

PS Position: NA.

Returned Parameters

Data String: Contents of the current presentation space.

Data String Length: Length of the data string returned.

Return Code: The following codes are valid:

Code	Explanation
0	The function was successful.
1	Not currently connected to a presentation space.
4	Successful, but host input available.
5	Successful, but keyboard locked.
9	A system error occurred.

Remarks

Returns 4-byte character/attributes as represented internally by the *teem* product emulator. The character/attribute value is emulation specific. Refer to the *Field Attribute Representation* appendix for details.

The IBM 5250 emulation supports a presentation space of 24 rows by 80 columns. When an error message is received from the host or when the operator presses the **SysReq** key, a 25th row is displayed. When row 25 is displayed, it is a valid area for this function.

COPY FORMATTED PRESENTATION SPACE TO STRING (108)

Copies all or part of the internal 4-byte representation of the specified portion of the currently connected formatted presentation space to a string.

Prerequisite Calls

Connect Presentation Space (1)

Supplied Parameters

Function Number: Must be 108.

Data String: Predefined buffer to hold string.

Data String Length: Length of data string required.

PS Position: Position in presentation space to start copy.

Returned Parameters

Data String: Contents of the current presentation space.

Data String Length: Length of the data string returned.

Return Code: The following codes are valid:

Code	Explanation
0	The function was successful.
1	Not currently connected to a presentation space.
2	Data string length 0 specified.
4	Successful, but host input available.
5	Successful, but keyboard locked.
7	Invalid presentation space position.
9	A system error occurred.

Remarks

Returns 4-byte character/attributes as represented internally by the *teem* product emulator. The character/attribute value is emulation specific. Refer to the *Field Attribute Representation* appendix for details.

The IBM 5250 emulation supports a presentation space of 24 rows by 80 columns. When an error message is received from the host or when the operator presses the **SysReq** key, a 25th row is displayed. When row 25 is displayed, it is a valid area for this function.

COPY OIA (13)

This function returns a copy of the status line (Operator Information Area) contents for the currently connected presentation space.

Prerequisite Calls

Connect Presentation Space (1)

Supplied Parameters

Function Number: Must be 13.

Data String: Predefined buffer to hold string.

Data String Length: Length of the target data string (in bytes).
For the IBM 3270 and 5250 emulations this is 103 bytes.

PS Position: NA.

Returned Parameters

Data String: Status line (OIA) data string (see **Remarks** section).

Data String Length: Length of data string returned (in bytes).

Return Code: The following codes are valid:

Code	Explanation
0	The function was successful.
1	Not currently connected to a presentation space.
2	Error in specifying data string length. No OIA data returned.
4	OIA data returned, but presentation space is busy.
5	OIA data returned, but presentation space locked.
9	A system error occurred.

Remarks

The status line must be enabled (active) for this function to work. The following sections describe the data string returned for the IBM 3270 emulation, the IBM 5250 emulation, then for the other emulations.

IBM 3270 Emulation

In the IBM 3270 emulation, the requested Data String Length must be 103, and the returned Data String contains the following data:

Offset 0	**Emulation Mode** (3270 = **1**, hexadecimal character 31).
Offset 1-80	**OIA Image**.
Offset 81-102	**OIA Group Indicators**.

The OIA image is a copy of the information on the displayed status line, in IBM 3270 PC format, using the 3270 Host Presentation Space characters shown below.

	0x	1x	2x	3x	4x	5x	6x	7x	8x	9x	Ax	Bx	Cx	Dx	Ex	Fx
x0	NUL	SP	0	&	à	ä	Á	Ä	a	q	A	Q			P	
x1	EM	=	1	–	è	ë	È	Ë	b	r	B	R	—		S	
x2	FF	’	2	.	ì	ï	Ì	ï	c	s	C	S			→	
x3	NL	”	3	,	ò	ö	Ò	Ö	d	t	D	T				
x4	STP	/	4	:	ù	ü	Ù	Ü	e	u	E	U				
x5	CR	\	5	+	ã	â	Ã	Â	f	v	F	V				
x6		¦	6	¬	õ	ê	Õ	Ê	g	w	G	W				
x7		¦	7	¯	ÿ	î	Y	Î	h	x	H	X				
x8	>	?	8	°	à	ô	A	Ô	i	y	I	Y	←		µ	¿
x9	<	!	9		è	û	E	Û	j	z	J	Z			²	
xA	[	$	ß	^	é	á	E	Á	k	æ	K	Æ			³	
xB	]	¢	§	~	ì	é	I	É	l	ø	L	Ø				
xC	)	£	#	¨	ò	í	O	Í	m	ª	M	º	A			a
xD	(	¥	@	`	ù	ó	U	Ó	n	ç	N	Ç	B		↔	
xE	}	Pts	%	´	ü	ú	Y	Ú	o	;	O	;				¡
xF	{		_		ç	ñ	C	Ñ	p	*	P	*				Not Supported

The OIA Group Indicators comprise bits representing the state of the connected session. Groups are divided by the host function they represent. The bits in each group are ordered so that the high-order bits represent the states of higher priority (bit 7 is high-order, hexadecimal 80). Therefore, if more than one state is active within a group, the state with the highest priority is the one represented in the OIA image for the session. The following list shows the meaning of the bits for each group indicator.

Group 1 (offset 81): Online and screen ownership

7-6 Reserved
5 SSCP-LU session owns screen
4 LU-LU session owns screen
3 Online and not owned
2 Subsystem ready
1-0 Reserved

Group 2 (offset 82): Character selection (not supported)

7-0 reserved

Group 3 (offset 83): Shift state

7 Reserved
6 Numeric
5-0 Reserved

Group 4 (offset 84): PSS group 1 (not supported)

7-0 Reserved

Group 5 (offset 85): Highlight group 1 (not supported)

7-0 Reserved

Group 6 (offset 86): Colour group 1 (not supported)

7-0 Reserved

Group 7 (offset 87): Insert

7 Insert mode
6-0 Reserved

Group 8 (offset 88-92): Input inhibited (5 bytes)

(offset 88)

7-4 Reserved
3 Program check
2-0 Reserved

(offset 89)

7 Device busy (OIA time)
6-4 reserved
3 Too much entered
2-0 Reserved

(offset 90)

7-4 Reserved
3 Wrong place
2-0 Reserved

(offset 91)

7-6 Reserved
5 System wait
4-0 Reserved

(offset 92)

7-0 Reserved

Group 9 (offset 93): PSS group 2 (not supported)

7-0 Reserved

Group 10 (offset 94): Highlight group 2 (not supported)

7-0 Reserved

Group 11 (offset 95): Colour group 2 (not supported)

7-0 Reserved

Group 12 (offset 96): Communication error reminder (not supported)

7-0 Reserved

Group 13 (offset 97): Printer status (not supported)

7-0 Reserved

Group 14 (offset 98): Graphics (not supported)

7-0 Reserved

Group 15 (offset 99): reserved

7-0 Reserved

Group 16 (offset 100): Autokey play/record status (not supported)

7-0 Reserved

Group 17 (offset 101): Autokey abort/pause status (not supported)

7-0 Reserved

Group 18 (offset 102): Enlarge state (not supported)

7-0 Reserved

IBM 5250 Emulation

In the IBM 5250 emulation, the requested Data String Length must be 103, and the returned Data String contains the following data:

Offset 0	**Emulation Mode** (5250 = **9**, hexadecimal character 39).
Offset 1-80	**OIA Image**.
Offset 81-102	**OIA Group Indicators**.

The OIA image is a copy of the information on the displayed status line.

The OIA Group Indicators comprise bits representing the state of the connected session. Groups are divided by the host function they represent. The bits in each group are ordered so that the high-order bits represent the states of higher priority (bit 7 is high-order, hexadecimal 80). Therefore, if more than one state is active within a group, the state with the highest priority is the one represented in the OIA image for the session. The following list shows the meaning of the bits for each group indicator.

Group 1 (offset 81): Online and screen ownership

7-5	Reserved
4	System available
3	Reserved
2	Subsystem ready
1-0	Reserved

Group 2 (offset 82): Character selection (not supported)

7-0	Reserved

Group 3 (offset 83): Shift state

7-0	Reserved

Group 4 (offset 84): PSS group 1 (not supported)

7-0	Reserved

Group 5 (offset 85): Highlight group 1 (not supported)

7-0	Reserved

Group 6 (offset 86): Colour group 1 (not supported)

7-0	Reserved

Group 7 (offset 87): Insert

7	Insert mode
6-0	Reserved

Group 8 (offset 88-92): Input inhibited (5 bytes)

(offset 88)

7-0	Reserved

(offset 89)

7-0 Reserved

(offset 90)

7-3 Reserved
2 Operator input error
1-0 Reserved

(offset 91)

7-6 Reserved
5 System wait
4-0 Reserved

(offset 92)

7-0 Reserved

Group 9 (offset 93): PSS group 2 (not supported)

7-0 Reserved

Group 10 (offset 94): Highlight group 2 (not supported)

7-0 Reserved

Group 11 (offset 95): Colour group 2 (not supported)

7-0 Reserved

Group 12 (offset 96): Communication error reminder (not supported)

7-1 Reserved
0 Message waiting

Group 13 (offset 97): Printer status (not supported)

7-0 Reserved

Group 14 (offset 98): Graphics (not supported)

7-0 Reserved

Group 15 (offset 99): reserved

7-0 Reserved

Group 16 (offset 100): Autokey play/record status (not supported)

7-0 Reserved

Group 17 (offset 101): Autokey abort/pause status (not supported)

7-0 Reserved

Group 18 (offset 102): Enlarge state (not supported)

7-0 Reserved

Other Emulations (except *teemtalk-1779W* & *teemX-1779*)

The size and status of the OIA reported is emulation dependent as listed in the table below.

Offset 0 **Emulation Mode**
Offset 1 **Status Type**
Offset 2-> **Status Line(s)**

Emulation	Mode	Status Type	Status Line(s)
VT52 **VT100** **VT320 (7-bit)** **VT320 (8-bit)** **DG200/410** **ANSI BBS**	0 1 2 3 7 9	0 - None 1 - Status 2 - Host Writable	None 80 character status 80/132 characters
Viewdata 40 **Viewdata 80** **Viewdata Split**	4 5 6	0 - None	None
HP700/92	8	0 - None 1 - Aids 2 - Modes 3 - User Keys 4 - Marg/Cols/Tabs 5 - User Keys Define 6 - Device Control 7 - Device Modes 8 - Host Writable	None 2 rows of 80 columns (types 1 - 8)
PT250	10	1 - Status 2 - System Error	80/132 character status 80/132 character error
TA6530	11	1 - Status 2 - Error	80 character status 80 character error
IBM 3151	13	0 - None 1 - Operator Message 2 - Host Writable 3 - PF Key	None 80/132 characters 80/132 characters 80/132 characters
Wyse	14 to 21	Lower 4 bits: 0 - Blank Status 1 - Standard Status 2 - Extended Status Upper 4 bits: 0 - No Label Line 1 - Unshifted Label 2 - Shifted Label	 80/132 characters 80/132 characters 80/132 characters 80/132 characters 80/132 characters 80/132 characters
AT&T 4410	22	1 - PF Keys	2 rows of 80 columns
Stratus V102	23	0 - None	None
ICL 7561	24	1 - Status	80 character status

teemtalk-1779W & teemX-1779

The size and status of the OIA reported is emulation dependent as listed in the table below.

Offset 0 **Operating Mode**
Offset 1 **Emulation Mode**
Offset 2 **Status Type**
Offset 3-> **Status Line(s)**

Operating Mode (offset 0):

- **a** DEC
- **b** IBM
- **c** ICL
- **d** HW
- **e** HPRATES
- **f** HPCAPPS
- **g** PRS 40

Emulation	Mode	Status Type	Status Line(s)
VT52 **VT100** **VT320 (7-bit)** **VT320 (8-bit)** **DG200/410** **ANSI BBS**	0 1 2 3 7 9	0 - None 1 - Status 2 - Host Writable	None 80 character status 80/132 characters
Viewdata 40 **Viewdata 80** **Viewdata Split**	4 5 6	0 - None	None
HP2392A	8	0 - None 1 - Aids 2 - Modes 3 - User Keys 4 - Marg/Cols/Tabs 5 - User Keys Define 6 - Device Control 7 - Device Modes 8 - Host Writable	None 2 rows of 80 columns (types 1 - 8)
IBM 3151	13	0 - None 1 - Operator Message 2 - Host Writable 3 - PF Key	None 80/132 characters 80/132 characters 80/132 characters
ICL 7561	24	1 - Status	80 character status

COPY PRESENTATION SPACE (5)

Copies the entire contents of the connected presentation space to a data string.

Prerequisite Calls

Connect Presentation Space (1)

Supplied Parameters

Function Number: Must be 5.

Data String: Predefined buffer, which must be at least the size of the current presentation space, or twice that if the EAB option is specified.

Data String Length: NA. The length of the presentation space in implied.

PS Position: NA.

Returned Parameters

Data String: Contents of the presentation space.

Data String Length: Length of the data string returned (in bytes), or twice that if the EAB option is specified.

Return Code: The following codes are valid:

Code	Explanation
0	The function was successful.
1	Not currently connected to a presentation space.
4	Successful, but host input available.
5	Successful, but keyboard locked.
9	A system error occurred.

Remarks

Returns 8-bit ASCII characters (ISO Latin-1).

This function is affected by the ATTRIB/NOATTRIB, EAB/NOEAB and the XLATE/NOXLATE options of the **Set Session Parameters** (9) function.

If you want to copy only part of the presentation space, use **Copy Presentation Space To String** (8).

The IBM 5250 emulation supports a presentation space of 24 rows by 80 columns. When an error message is received from the host or when the operator presses the **SysReq** key, a 25th row is displayed. When row 25 is displayed, it is a valid area for this function.

COPY PRESENTATION SPACE TO STRING (8)

Copies all or part of the contents of the connected presentation space to a data string.

Prerequisite Calls

Connect Presentation Space (1)

Supplied Parameters

Function Number: Must be 8.

Data String: Predefined buffer to hold string (twice the length requested if the EAB option has been specified).

Data String Length: Length of data string required.

PS Position: Position in the presentation space to start the copy. This is the offset into the presentation space where position 1 is the top left corner of the screen display (row 1 and column 1), and the highest position number is the bottom right corner.

Returned Parameters

Data String: Contents of the current presentation space as requested.

Data String Length: Length of the data string returned in bytes.

Return Code: The following codes are valid:

Code	Explanation
0	The function was successful.
1	Not currently connected to a presentation space.
2	Data string length 0, or PS position + (length - 1) greater than PS size.
4	Successful, but host input available.
5	Successful, but keyboard locked.
7	Invalid presentation space position.
9	A system error occurred.

Remarks

Returns 8-bit ASCII characters (ISO Latin-1). This function is affected by the ATTRIB/NOATTRIB, EAB/NOEAB and the XLATE/NOXLATE options of the **Set Session Parameters** (9) function.

The IBM 5250 emulation supports a presentation space of 24 rows by 80 columns. When an error message is received from the host or when the operator presses the **SysReq** key, a 25th row is displayed. When row 25 is displayed, it is a valid area for this function.

COPY STRING TO FIELD (33)

Transfers a string of characters to a field in the currently connected presentation space.

Prerequisite Calls

Connect Presentation Space (1)

Supplied Parameters

Function Number: Must be 33.

Data String: String containing 8-bit ASCII displayable characters for the target field in the presentation space.

Data String Length: Length of data string (in bytes). This parameter is not applicable if in end of transmission (EOT) mode (see **Set Session Parameters** (9)).

PS Position: Any character position within the target field. (Note that the copy will always start at the beginning of the field.)

Returned Parameters

Return Code: The following codes are valid:

Code	Explanation
0	The function was successful.
1	Not currently connected to a presentation space.
2	Data string length 0 specified.
4	Function inhibited as target presentation space was busy.
5	Target field was protected or inhibited, or invalid data was sent.
6	Copy complete but data truncated.
7	Presentation space position invalid.
9	A system error occurred.
24	The presentation space is unformatted.

Remarks

This function is not supported by VT52, ANSI-BBS, DG410, AT&T 4410, Stratus V102, or the Viewdata modes. It is only valid when the presentation space is field formatted. Field formatting is emulation dependent as shown in the table on page 3-2.

Only 8-bit ASCII displayable characters are supported. An attempt to write non-displayable characters (e.g. new line, carriage return or attribute values) will return error code **5** (invalid data was sent); the string will be copied up to but not including the invalid character.

The location of a character position within the target field is determined by the offset into the presentation space where position 1 is the top left corner of the screen display (row 1 and column 1), and the highest position number is the bottom right corner.

The first byte of the data string is always placed in the first character position of the target field. If the data string provided is longer than the target field, the data string is truncated to fit and a return code **6** (copy complete but data truncated) is returned.

End of transmission (EOT) mode (see **Set Session Parameters** (9)) is supported for all emulations.

The IBM 5250 emulation supports a presentation space of 24 rows by 80 columns. When an error message is received from the host or when the operator presses the **SysReq** key, a 25th row is displayed. When row 25 is displayed, it is a valid area for this function.

COPY STRING TO PRESENTATION SPACE (15)

Transfers characters to a specified position in the currently connected presentation space.

Prerequisite Calls

Connect Presentation Space (1)

Supplied Parameters

Function Number: Must be 15.

Data String: String containing 8-bit ASCII characters to be copied to the presentation space.

Data String Length: Length of the data string (in bytes). This parameter is not applicable if in end of transmission (EOT) mode (see **Set Session Parameters** (9)).

PS Position: Position in the presentation space to start the copy (for formatted presentation spaces only, otherwise this parameter is ignored).

Returned Parameters

Return Code: The following codes are valid:

Code	Explanation
0	The function was successful.
1	Not currently connected to a presentation space.
2	Data string length 0 specified.
4	Function inhibited as target presentation space was busy.
5	Target field was protected or inhibited, or invalid data was sent.
7	Presentation space position invalid.
9	A system error occurred.

Remarks

This function is faster than **Send Keystring** (3) and **Send Keypress** (103), but these functions provide additional support for special characters. The result is very similar to that achieved by **Send Keystring** (3), but bypasses the keyboard input checks for special keystrokes and macro expansion.

The PS position is determined by the offset into the formatted presentation space where position 1 is the top left corner of the screen display (row 1 and column 1), and the highest position number is the bottom right corner.

In unformatted/non-block mode presentation spaces, the characters will be sent to the host computer (unless it is in local mode), not to the screen.

End of transmission (EOT) mode (see **Set Session Parameters** (9)) is supported for all emulations.

The IBM 5250 emulation supports a presentation space of 24 rows by 80 columns. When an error message is received from the host or when the operator presses the **SysReq** key, a 25th row is displayed. When row 25 is displayed, it is a valid area for this function.

DISCONNECT PRESENTATION SPACE (2)

teemtalk

Drops the connection between the API application program and the host presentation space. It also cancels the effect of the **Reserve** (11) function.

*teem*X

As above if an API configuration file was used, otherwise this disconnects from the current presentation space by terminating that instance of *teem*X.

Prerequisite Calls

Connect Presentation Space (1)

Supplied Parameters

Function Number: Must be 2.

Data String: NA.

Data String Length: NA.

PS Position: NA.

Returned Parameters

Return Code: The following codes are valid:

Code	Explanation
0	***teemtalk:*** The function was successful. ***teemX:*** The function was successful; the presentation space is disconnected.
1	Not currently connected to a presentation space.
8	Either USENEW is set and there is an existing session, or USEOLD is set and there isn't an existing session. (See the **Set Session Parameters** (9) function.)
9	A system error occurred.

Remarks

This function will cause **Stop Host Notification** (25) and **Stop Keystroke Intercept** (53) to be actioned if the relevant '**Start**' function has been called since the **Connect Presentation Space** (1) for this presentation space.

This function does not reset the session parameters to their default condition. Your API application must call the **Reset System** (21) function to reset to the defaults.

There is no need to disconnect from all instances of the *teemtalk* or *teem*X product before exiting, but they will remain functional after the API application terminates.

After the function is actioned, the presentation space is available for use by a different application (except *teem*X if no API configuration file was used).

FIND FIELD LENGTH (32)

Returns the length of a target field in a field formatted presentation space. The function is valid for protected and unprotected fields. Note that you can use the **Find Field Position** (31) function to locate a particular field.

Prerequisite Calls

Connect Presentation Space (1)

Supplied Parameters

Function Number: Must be 32.

Data String: Two character field code as follows:

Code	Explanation
[space][space]	Current field.
T[space]	Current field.
N[space]	The Next field (protected or not).
NP	The Next Protected field.
NU	The Next Unprotected field.
P[space]	The Previous field (protected or not).
PP	The Previous Protected field.
PU	The Previous Unprotected field.

Data String Length: NA (2 implied).

PS Position: Any character position within the target field.

Returned Parameters

Data String Length:

0 If the return code is **28**, the field length is zero. If the return code is **24**, the presentation space is unformatted.

>0 The number of characters from the beginning of the specified field up to the character preceding the next attribute byte.

Return Code: The following codes are valid:

Code	Explanation
0	The function was successful.
1	Not currently connected to a presentation space.
2	Parameter error.
7	Invalid presentation space specified.
9	A system error occurred.
24	The presentation space is unformatted, or no such field found.
28	Field length was **0**.

Remarks

This function is not supported by VT52, ANSI-BBS, DG410, AT&T 4410, Stratus V102, or the Viewdata modes. It is only valid when the presentation space is field formatted. Field formatting is emulation dependent as shown in the table on page 3-2.

Use the **Find Field Position** (31) function to locate a particular field. The location of a character position within the target field is determined by the offset into the presentation space where position 1 is the top left corner of the screen display (row 1 and column 1), and the highest position number is the bottom right corner.

The IBM 5250 emulation supports a presentation space of 24 rows by 80 columns. When an error message is received from the host or when the operator presses the **SysReq** key, a 25th row is displayed. When row 25 is displayed, it is a valid area for this function.

FIND FIELD POSITION (31)

Returns the start position of a target field in a field-formatted presentation space. The function can be used for protected or unprotected fields.

Prerequisite Calls

Connect Presentation Space (1)

Supplied Parameters

Function Number: Must be 31.

Data String: Two character field code as follows:

Code	Explanation
[space][space]	Current field.
T[space]	Current field.
N[space]	The Next field (protected or not).
NP	The Next Protected field.
NU	The Next Unprotected field.
P[space]	The Previous field (protected or not).
PP	The Previous Protected field.
PU	The Previous Unprotected field.

Data String Length: NA (2 implied).

PS Position: Position within the presentation space to start the find.

Returned Parameters

Data String Length: Location of the first character position within the requested field, or **0** for return codes **24** and **28**.

Return Code: The following codes are valid:

Code	Explanation
0	The function was successful.
1	Not currently connected to a presentation space.
2	Parameter error.

Code	Explanation
7	Invalid presentation space specified.
9	A system error occurred.
24	The presentation space is unformatted, or no such field found.
28	Field length was **0**.

Remarks

This function is not supported by VT52, ANSI-BBS, DG410, AT&T 4410, Stratus V102, or the Viewdata modes. It is only valid when the presentation space is field formatted. Field formatting is emulation dependent as shown in the table on page 3-2.

Presentation space positions are determined by the offset into the presentation space where position 1 is the top left corner of the screen display (row 1 and column 1), and the highest position number is the bottom right corner.

The IBM 5250 emulation supports a presentation space of 24 rows by 80 columns. When an error message is received from the host or when the operator presses the **SysReq** key, a 25th row is displayed. When row 25 is displayed, it is a valid area for this function.

GET HIGHLIGHT AREA (132)

This will report the type and position of the highlighted area on the *teemtalk* screen, if any.

Prerequisite Calls

Connect Presentation Space (1)

Supplied Parameters

Function Number: Must be 132.

Data String: NA.

Data String Length: NA.

PS Position: NA.

Returned Parameters

Data String: If successful, contains the following data:

Offset	Definition
0	Area type as follows: **T** = for text area. **R** = for rectangular area.
1->	Start PS position (ASCII) **:** end PS position (ASCII).

Data String Length: If successful, length of the data string returned, else **0**.

Return Code: The following codes are valid:

Code	Explanation
0	The function was successful.
1	Not currently connected to a presentation space.
9	A system error occurred.
24	No area highlighted.

GET KEY (51)

This will receive a keystroke from a session that has keystroke intercept enabled (see **Start Keystroke Intercept** (50)) so that it can either be processed, accepted or rejected by your API program. This function can be used to intercept a string by placing it in a loop.

Prerequisite Calls

Start Keystroke Intercept (50)

Supplied Parameters

Function Number: Must be 51.

Data String: Contains the following data:

Offset	Definition
0	One character ID for presentation space, or blank for current presentation space.
1-5	Blanks for holding symbol representation of keystroke.
6-7	Reserved.

Data String Length: NA (8 implied).

PS Position: NA.

Returned Parameters

Data String: Contains the following data:

Offset	Definition
0	One character ID for presentation space, or blank for current presentation space.
1	Character code as follows: **A** = for ASCII character. **M** = for keystroke mnemonic. **S** = for special shift (**Alt**/**Ctrl**) returned with other data.
2-7	Keystroke data.

In the following returned data string examples, the escape character is represented by the **@** symbol. Any ASCII keystroke character can be specified as the escape character for keystroke mnemonics by using the **ESC=** option of the **Set Session Parameters** (9) function. Refer also to the *ASCII Mnemonics* section in the *Overview* chapter for a list of all supported mnemonics and their meaning.

BAg Indicates that the API program is connected to presentation space **B**, the keystrokes are ASCII (**A**), and the returned key is the character **g**.

BM@4 Indicates that the API program is connected to presentation space **B**, the keystrokes are mnemonic (**M**), and the returned key is **PF4**.

BS@A@4 Indicates that the API program is connected to presentation space **B**, the keystrokes are in special shift state (**S**), and the returned key is **Alt** + **PF4**.

BS@r@4 Indicates that the API program is connected to presentation space **B**, the keystrokes are in special shift state (**S**), and the returned key is **Ctrl** + **PF4**.

Data String Length: Length of the data string returned.

Return Code: The following codes are valid:

Code	Explanation
0	The function was successful.
1	An invalid presentation space was specified.
5	The AID keys only option was specified by **Start Keystroke Intercept** (50) and non-AID keys are inhibited by this session type.
8	**Start Keystroke Intercept** (50) has not been called.
9	A system error occurred.
20	An undefined key combination was typed.
25	The requested keystrokes were not available in the input queue.
31	Keystrokes were lost due to input queue overflow.

Remarks

The **Start Keystroke Intercept** (50) function specifies the size of the keystroke intercept buffer in which keystrokes are queued asynchronously. Each keystroke occupies three bytes in the buffer. If **Get Key** returns code **31**, either increase the size

of the keystroke intercept buffer, or execute **Get Key** more frequently to retrieve keystrokes from the buffer so that space is made available.

Get Key will read keystrokes into the data area of your application. A WAIT option can be specified by **Set Session Parameters** (9) so that control is not returned to the API program until **Get Key** has intercepted a keystroke as defined by **Start Keystroke Intercept** (50).

If only AID keys (e.g. **Enter**) are required, as may be the case for field formatted sessions, set the **Start Keystroke Intercept** (50) option code to **D**.

The **Send Keystring** (3) function can be used to pass original keystrokes to the host-connected presentation space.

The type of keystroke data received by the host application when an operator types a key or shift key combination is as follows:

Key defined as an ASCII character

A 1-byte ASCII value corresponding to that character is received.

Key defined as a function

A 2-, 4-, or 6-byte ASCII mnemonic corresponding to that function is received. For example, if the **Delete** key is pressed, then **@D** is received.

Defined shift key combination

An ASCII character or the 2-, 4-, or 6-byte ASCII mnemonic corresponding to that function is received. For example, if the combination **Alt** + **z** is defined as delete, then **@D** is received when **Alt** + **z** is typed.

Single undefined key

Nothing is received. **Get Key** returns a return code **20**.

Undefined shift key combination

Depends on whether the unshifted state of the key is defined.

If the unshifted state is not defined, nothing is received. **Get Key** returns a return code **20**.

If the unshifted state is defined as an ASCII character, the ASCII mnemonic for the shift key followed by the character associated with the unshifted key is received. For example, if the **Ctrl** key is pressed with a key defined as lowercase **f**, **@rf** will be received.

If the unshifted state is defined as afunction, the ASCII mnemonic for the shift key followed by the ASCII mnemonic corresponding to the defined function is received. For example, if the **p** key is mapped as Print Screen, and **Ctrl** + **p** is undefined, when **Ctrl** + **p** is pressed, **@r@A@T** (the mnemonics for Ctrl and Print Screen) will be received, not **@rp** (the mnemonics for Ctrl and p).

GET TEEMTALK / TEEMX PARAMETER (110)

Allows current *teemtalk* or *teem*X settings to be examined.

Prerequisite Calls

Connect Presentation Space (1)

Supplied Parameters

Function Number: Must be 110.

Data String: Predefined buffer to hold string.

Data String Length: Length of the data string (in bytes). This parameter is not applicable if in end of transmission (EOT) mode (see **Set Session Parameters** (9)).

PS Position: NA.

Returned Parameters

Data String: String containing the returned value.

Data String Length: Length of the data string returned.

Return Code: The following codes are valid:

Code	Explanation
0	The function was successful.
1	Not currently connected to a presentation space.
2	Parameter error - unsuccessful.
9	A system error occurred.

Remarks

The valid *teemtalk* or *teem*X settings are those listed for the script language PSET command as documented in the *teem* product Programmer's Guide.

End of transmission (EOT) mode (see **Set Session Parameters** (9)) is supported for all emulations.

GET WINDOW HANDLE (130)

Returns the window handle for the current connection.

Prerequisite Calls

Connect Presentation Space (1)

Supplied Parameters

Function Number: Must be 130.

Data String: Predefined buffer to hold string.

Data String Length: NA.

PS Position: NA.

Returned Parameters

Data String: String containing the numeric value of the window handle.

Data String Length: Length of the data string returned.

Return Code: The following codes are valid:

Code	Explanation
0	The function was successful.
1	Not currently connected to a presentation space.
9	A system error occurred.

HOST INPUT (129)

Imitates a block of input data from the host computer.

Prerequisite Calls

Connect Presentation Space (1)

Supplied Parameters

Function Number: Must be 129.

Data String: Block of data to imitate data from the host computer.

Data String Length: Length of the data string (in bytes). This parameter is not applicable if in end of transmission (EOT) mode (see **Set Session Parameters** (9)).

PS Position: NA.

Returned Parameters

Return Code: The following codes are valid:

Code	Explanation
0	The function was successful.
1	Not currently connected to a presentation space.
4	Host session was busy (*after* data processed).
5	Input was inhibited (*after* data processed).
9	A system error occurred.

Remarks

This function can be used to imitate host activity.

PAUSE (18)

This causes the application to wait for a specified amount of time or until a host event occurs if **Start Host Notification** (23) has been called.

Prerequisite Calls

Connect Presentation Space (1)

Supplied Parameters

Function Number: Must be 18.

Data String: One character ID of the required presentation space, or blank for all presentation spaces that have **Start Host Notification** (23) enabled (if required for IPAUSE).

Data String Length: Pause duration in half-second increments. For example, a value of 120 will result in a pause duration of 60 seconds.

PS Position: NA.

Returned Parameters

Return Code: The following codes are valid:

Code	Explanation
0	The wait duration has expired.
9	A system error occurred.
26	Presentation space or status line updated.

Remarks

The behaviour of this function depends on the session parameters FPAUSE and IPAUSE (see **Set Session Parameters** (9)).

If **Start Host Notification** (23) is not enabled, or the FPAUSE option is set in **Set Session Parameters** (9) (default), the **Pause** function will wait for the specified time and return code **0**.

If the IPAUSE option is set, **Pause** will return code **26** if the presentation space or status line has been updated. Determine this by **Query Host Update** (24) before calling **Pause** again, otherwise it will return code **26** continually.

PERICOM FUNCTIONS (128)

This function provides additional Pericom private functions.

Prerequisite Calls

Connect Presentation Space (1)

Supplied Parameters

Function Number: Must be 128.

Data String: Depends on the Pericom function.

Data String Length: Depends on the Pericom function.

PS Position: Pericom function number (refer to the function descriptions below).

Returned Parameters

Return Code: The following codes are valid:

Code	Explanation
0	The function was successful.
1	Not currently connected to a presentation space.
2	Parameter error - unsuccessful.
8	No prerequisite call.
9	A system error occurred.

Functions

SAVE SETTINGS (1)

Saves the current *teemtalk* or *teem*X settings.

Supplied Parameters

Data String: NA

Data String Length: NA

POST INTERCEPT STATUS (52)

This informs the emulation that a keystroke obtained by **Get Key** (51) has been accepted or rejected, and will cause a warning beep to sound if a keystroke is rejected.

Prerequisite Calls

Start Keystroke Intercept (50)

Supplied Parameters

Function Number: Must be 52.

Data String: A 2-byte data string containing the following:

Offset	Definition
0	One character ID for presentation space, or blank for current presentation space.
1	**A** = for keystroke accepted. **R** = for keystroke rejected.

Data String Length: NA (2 implied).

PS Position: NA.

Returned Parameters

Return Code: The following codes are valid:

Code	Explanation
0	The function was successful.
1	An invalid presentation space was specified.
2	An invalid session option was specified.
8	**Start Keystroke Intercept** (50) has not been called for this presentation space.
9	A system error occurred.

QUERY CURSOR LOCATION (7)

This function indicates the current cursor position in the presentation space.

Prerequisite Calls

Connect Presentation Space (1)

Supplied Parameters

Function Number: Must be 7.

Data String: NA.

Data String Length: NA.

PS Position: NA.

Returned Parameters

Data String Length: Presentation space positional value of the cursor.

Return Code: The following codes are valid:

Code	Explanation
0	The function was successful.
1	Not currently connected to a presentation space.
9	A system error occurred.

Remarks

To translate the presentation space positional value into a cursor location in rows and columns, use **Convert Position/RowCol** (99).

The IBM 5250 emulation supports a presentation space of 24 rows by 80 columns. When an error message is received from the host or when the operator presses the **SysReq** key, a 25th row is displayed. When row 25 is displayed, it is a valid area for this function.

QUERY FIELD ATTRIBUTE (14)

Provides a copy of the attribute byte of the field containing the specified presentation space position.

Prerequisite Calls

Connect Presentation Space (1)

Supplied Parameters

Function Number: Must be 14.

Data String: NA.

Data String Length: NA.

PS Position: PS position of any byte in the target field.

Returned Parameters

Data String: NA.

Data String Length: If the screen is formatted and the emulation supports attribute bytes then the data string length contains the attribute byte as specified in the *Field Attribute Representation* appendix.

If the screen is unformatted or the emulation does not support attribute bytes then **0** is returned.

Return Code: The following codes are valid:

Code	Explanation
0	The function was successful.
1	Not currently connected to a presentation space.
7	Presentation space position invalid.
9	A system error occurred.
24	Unformatted presentation space or attribute bytes not supported by the emulation.

Remarks

This function is not supported by VT52, ANSI-BBS, DG410, AT&T 4410, Stratus V102, or the Viewdata modes. It is only valid when the presentation space is field formatted. Field formatting is emulation dependent as shown in the table on page 3-2.

Presentation space positions are determined by the offset into the presentation space where position 1 is the top left corner of the screen display (row 1 and column 1), and the highest position number is the bottom right corner.

The current field attributes are determined by the bit setting of the attribute byte. Refer to the *Field Attribute Representation* appendix for a description of how field attributes are represented in the data string returned by this function.

See the **Query Formatted Field Attribute** (114) function for a more comprehensive function that returns the entire internally stored 4-byte field attribute.

QUERY FORMATTED FIELD ATTRIBUTE (114)

Provides a copy of the internal 4-byte representation of the attribute of the field containing the specified presentation space position.

Prerequisite Calls

Connect Presentation Space (1)

Supplied Parameters

Function Number: Must be 114.

Data String: Predefined buffer to hold string.

Data String Length: NA (4 bytes implied).

PS Position: PS position of any byte in the target field.

Returned Parameters

Data String: 4 bytes of character/attribute value.

Data String Length: Returns **4** if successful, otherwise **0**.

Return Code: The following codes are valid:

Code	Explanation
0	The function was successful.
1	Not currently connected to a presentation space.
7	Presentation space position invalid.
9	A system error occurred.
24	Unformatted presentation space.

Remarks

This function is not supported by VT52, ANSI-BBS, DG410, AT&T 4410, Stratus V102, or the Viewdata modes. It is only valid when the presentation space is field formatted. Field formatting is emulation dependent as shown in the table on page 3-2.

Presentation space positions are determined by the offset into the presentation space where position 1 is the top left corner of the screen display (row 1 and column 1), and the highest position number is the bottom right corner.

The character/attribute value is emulation specific. Refer to the *Field Attribute Representation* appendix for details.

QUERY HOST UPDATE (24)

Determines whether the specified presentation space and/or the OIA (status line) has been updated since the last call to **Start Host Notification** (23) or **Query Host Update** (24).

Prerequisite Calls

Start Host Notification (23)

Supplied Parameters

Function Number: Must be 24.

Data String: One character ID for presentation space, or blank for the current presentation space.

Data String Length: NA (1 implied).

PS Position: NA.

Returned Parameters

Return Code: The following codes are valid:

Code	Explanation
0	No updates since last call.
1	Invalid presentation space specified.
8	No prior **Start Host Notification** (23) for presentation space.
9	A system error occurred.
21	OIA updated.
22	Presentation space updated.
23	Presentation space and OIA updated.

Remarks

The presentation space does not have to be connected for this function to work, but a **Start Host Notification** (23) must have been issued to it beforehand.

If the presentation space has been terminated by the user (not by the API application) then return code **9** (system error) will be returned.

This function only supports reporting for both presentation space and OIA updates.

QUERY SESSION STATUS (22)

This function obtains session-specific information for a connected presentation space.

Prerequisite Calls

None.

Supplied Parameters

Function Number: Must be 22.

Data String: One character ID for presentation space, or blank for the current presentation space, plus 17 bytes for returned data.

Data String Length: 18 bytes.

PS Position: NA.

Returned Parameters

Data String: Contains the following returned data:

Offset	Definition
0	One character ID for presentation space.
1-8	Session long name (same as PSID).
9	D = IBM 3270 Display E = IBM 3270 Printer F = IBM 5250 Display T = Otherwise
10	0 (Hex 80 = IBM 3270 extended attribute supported).
11-12	Number of rows in presentation space (binary).
13-14	Number of columns in presentation space (binary)
15-16	IBM 3270 only: Host code page (binary).
17	Emulation mode (or operating mode for *teemtalk-1779W*, *teem*X-*1779*) as specified for **Copy OIA** (13).

Return Code: The following codes are valid:

Code	Explanation
0	The function was successful.
1	Invalid presentation space specified.
2	Invalid data string length.
9	A system error occurred.

Remarks

The number of rows and columns is zero for printer sessions.

QUERY SESSIONS (10)

This function returns details of all configured sessions.

Prerequisite Calls

None.

Supplied Parameters

Function Number: Must be 10.

Data String: Predefined buffer to hold string.

Data String Length: Minimum 12 times number of configured sessions.

PS Position: NA.

Returned Parameters

Data String: Contains the following 12 bytes of returned data for each configured session:

Offset	Definition
0	One character ID for presentation space.
1-8	Session long name (same as PSID).
9	Session type. (H = Host)
10-11	Size of presentation space (binary).

Data String Length: Number of configured sessions.

Return Code: The following codes are valid:

Code	Explanation
0	The function was successful.
2	Invalid data string length.
9	A system error occurred.

Remarks

If the return code is **0** or **2**, the data string length will indicate the number of configured sessions. This number can be used to determine the minimum data string length required.

teemtalk: A configured session is any session defined in the **TTHLLAPI.INI** file.

***teem*X:** A configured session is any session defined in the API configuration file, or any currently running instance of *teem*X previously launched by the API application and has not subsequently terminated.

The size of the presentation space is zero for printer sessions.

QUERY SOM LOCATION (133)

This function indicates the current position of the SOM (Start Of Message) indicator in the presentation space when running the ICL 7561 emulation.

Prerequisite Calls

Connect Presentation Space (1)

Supplied Parameters

Function Number: Must be 133.

Data String: NA.

Data String Length: NA.

PS Position: NA.

Returned Parameters

Data String Length: Presentation space positional value of the SOM indicator.

Return Code: The following codes are valid:

Code	Explanation
0	The function was successful.
1	Not currently connected to a presentation space.
4	Function inhibited - the ICL 7561 emulation is not running.
9	A system error occurred.

Remarks

To translate the presentation space positional value into a SOM indicator location in rows and columns, use **Convert Position/RowCol** (99).

QUERY SYSTEM (20)

Determines the level of API and *teem* product support as well as other system related values.

Prerequisite Calls

None.

Supplied Parameters

Function Number: Must be 20.

Data String: Predefined buffer to hold 35 bytes.

Data String Length: NA (implied length is 35 bytes).

PS Position: NA.

Returned Parameters

Data String: Contains the following 35 bytes:

Offset	Definition
0	API version number.
1-2	API level number.
3-8	API date (MMDDYY).
9-11	Reserved.
12	Hardware ***teemtalk:*** **A** = Personal Computer ***teemX:*** **U** = UNIX
13	Program type ***teemtalk:*** **P** = IBM PC 3270 ***teemX:*** **X** = X Windows
14-15	Reserved.
16-17	Emulator version for last connect, else **?**.
18	Emulator type: **2** = 320, **4** = 340, **M** = 1779, for last connect, else **?**.
19-24	Emulator date (YYMMDD) for last connect, else **?**.
25 -34	Reserved.

Data String Length: Returns **35** if successful, else **0**.

Return Code: The following codes are valid:

Code	Explanation
0	The function was successful.
9	A system error occurred.

RELEASE (12)

Unlocks the currently connected presentation space previously locked by the **Reserve** (11) function.

Prerequisite Calls

Connect Presentation Space (1)

Supplied Parameters

Function Number: Must be 12.

Data String: NA.

Data String Length: NA.

PS Position: NA.

Returned Parameters

Return Code: The following codes are valid:

Code	Explanation
0	The function was successful.
1	Not currently connected to a presentation space.
9	A system error occurred.

Remarks

If you do not release the presentation space, it will remained locked until a **Disconnect Presentation Space** (2) or **Reset System** (21) is called.

If a presentation space is reserved when a API application is terminated, the *teem* product will be unable to process user input.

RESERVE (11)

This function locks the currently connected presentation space from change by the user via the keyboard or mouse.

Prerequisite Calls

Connect Presentation Space (1)

Supplied Parameters

Function Number: Must be 11.

Data String: NA.

Data String Length: NA.

PS Position: NA.

Returned Parameters

Return Code: The following codes are valid:

Code	Explanation
0	The function was successful.
1	Not currently connected to a presentation space.
9	A system error occurred.

Remarks

Use the **Release** (12) function to unlock the presentation space. If you do not release the presentation space, it will remained locked until a **Disconnect Presentation Space** (2) or **Reset System** (21) is called.

If a presentation space is reserved when a API application is terminated, the *teem* product will be unable to process user input.

RESET SYSTEM (21)

This function reinitializes API to its starting state by defaulting all parameters and disconnecting any active sessions.

Prerequisite Calls

None.

Supplied Parameters

Function Number: Must be 21.

Data String: NA.

Data String Length: NA.

PS Position: NA.

Returned Parameters

Return Code: The following codes are valid:

Code	Explanation
0	The function was successful.
9	A system error occurred.

Remarks

This function disconnects all presently active instances of the *teem* product.

It is advisable to call this function prior to terminating the API application. This will ensure all copies of *teemtalk* or *teem*X are 'disconnected'.

SEARCH FIELD (30)

This function examines a field within a field-formatted presentation space for the occurence of a specified string. If the string is found, the presentation space position of the first character is returned. The function can be used for protected or unprotected fields.

Prerequisite Calls

Connect Presentation Space (1)

Supplied Parameters

Function Number: Must be 30.

Data String: Target string to search for in the presentation space (8-bit ASCII).

Data String Length: Length of the data string (in bytes). This parameter is not applicable in end of transmission (EOT) mode (see **Set Session Parameters** (9)).

PS Position: Position in the presentation space to start the search.

Returned Parameters

Data String Length: Presentation space position of the first character in the required data string, or **0** if not found.

Return Code: The following codes are valid:

Code	Explanation
0	The function was successful.
1	Not currently connected to a presentation space.
2	Parameter error.
7	Invalid presentation space specified.
9	A system error occurred.
24	The presentation space is unformatted or string not found.

Remarks

This function is not supported by VT52, ANSI-BBS, DG410, AT&T4410, Stratus V102, or the Viewdata modes. It is only valid when the presentation space is field formatted. Field formatting is emulation dependent as shown in the table on page 3-2.

Presentation space positions are determined by the offset into the presentation space where position 1 is the top left corner of the screen display (row 1 and column 1), and the highest position number is the bottom right corner.

The exact nature of the search within the field is determined by the following **Set Session Parameters** (9) options:

SRCHALL, SRCHFRWD	Searches the entire field that contains the PS position from the beginning to the end. The first instance of the required string is returned if it exists.
SRCHALL, SRCHBKWD	Searches the entire field that contains the PS position from the end to the beginning. The last instance of the required string is returned if it exists.
SRCHFROM, SRCHFRWD	The search begins at the PS position and continues to the end of the field until the specified string is found.
SRCHFROM, SRCHBKWD	The search begins at the PS position and continues to the start of the field until the specified string is found.

An attempt to search for non-displayable characters will not be successful and a return code **24** (string not found) will be issued.

SEARCH PRESENTATION SPACE (6)

Examines the connected presentation space for the occurrence of a specified string (characters only). If the string is found, the presentation space position of the first character is returned.

This function can be used to determine when the session is ready for input by searching for a required prompt.

Prerequisite Calls

Connect Presentation Space (1)

Supplied Parameters

Function Number: Must be 6.

Data String: Target string to search for in the presentation space (8-bit ASCII).

Data String Length: Length of the data string (in bytes). This parameter is not applicable in end of transmission (EOT) mode.

PS Position: Position in the presentation space to start the search if SRCHFROM is set by the **Set Session Parameters** (9) function, else NA.

Returned Parameters

Data String Length: Presentation space position of the first character in the required data string, or **0** if not found.

Return Code: The following codes are valid:

Code	Explanation
0	The function was successful.
1	Not currently connected to a presentation space.
2	Parameter error.
7	Host presentation space position invalid.
9	A system error occurred.
24	String not found.

Remarks

Presentation space positions are determined by the offset into the presentation space where position 1 is the top left corner of the screen display (row 1 and column 1), and the highest position number is the bottom right corner.

The exact nature of the search within the field is determined by the following **Set Session Parameters** (9) options:

SRCHALL, SRCHFRWD	Ignores the specified PS position and searches the entire presentation space from the beginning to the end. The first instance of the required string is returned if it exists.
SRCHALL, SRCHBKWD	Ignores the specified PS position and searches the entire presentation space from the end to the beginning. The last instance of the required string is returned if it exists.
SRCHFROM, SRCHFRWD	The search begins at the PS position and continues to the end of the presentation space until the specified string is found.
SRCHFROM, SRCHBKWD	The search begins at the PS position and continues to the start of the presentation space until the specified string is found.

An attempt to search for non-displayable characters will not be successful and a return code **24** (string not found) will be issued.

SEND KEYPRESS (103)

Imitates the pressing of an emulator macro key by the user (in *teem* product internal format).

Prerequisite Calls

Connect Presentation Space (1)

Supplied Parameters

Function Number: Must be 103.

Data String: Macro key number or virtual key name (refer to the *teem* product User's Guide) as an 8-bit ASCII string.

Data String Length: Length of the data string. This parameter is not applicable if in end of transmission (EOT) mode.

PS Position: NA.

Returned Parameters

Return Code: The following codes are valid:

Code	Explanation
0	The function was successful.
1	Not currently connected to a presentation space.
2	Parameter error.
4	Host session busy.
5	Input was inhibited.
9	A system error occurred.

Remarks

This function simulates the pressing of a single emulator macro key (the macro key numbers and virtual key names are defined in the *teem* product User's Guide). In most cases this can also be achieved using **Send Keystring** (3).

If NORESET is requested by **Set Session Parameters** (9), a reset function is performed before the macro key. This will reset all states that can be reset except input-inhibited states. Note that soft keyboard locks will also be reset.

The effect on the emulator will be exactly as if the keystroke had occurred at the keyboard. If the keyboard is locked then a bell will sound. If the emulation is busy then the key macro will be placed on the keyboard queue and processed when the emulator becomes free.

The emulator will return once the keystroke string has been actioned, input was inhibited or host session was busy whenever these two conditions exist.

Error code **4** (host session busy) will be returned when the host is busy and the keyboard is locked.

SEND KEYSTRING (3)

Imitates the pressing of a series of keys by the user.

Prerequisite Calls

Connect Presentation Space (1)

Supplied Parameters

Function Number: Must be 3.

Data String: Series of keystrokes (maximum 255 bytes) specified as ASCII characters, virtual key names as listed in the *teem* product User's Guide, or ASCII mnemonics (refer to the *Overview* chapter).

Data String Length: Length of the data string. This parameter is not applicable if in end of transmission (EOT) mode.

PS Position: NA.

Returned Parameters

Return Code: The following codes are valid:

Code	Explanation
0	The function was successful.
1	Not currently connected to a presentation space.
2	Parameter error.
4	Host session was busy.
5	Input was inhibited.
9	A system error occurred.

Remarks

Keystrokes which cannot be represented in ASCII (8-bit) characters (e.g. to represent hitting a cursor key) may be specified using virtual key names (e.g. **<VK_LEFT>** for cursor left).

Function keys can be represented by ASCII mnemonics as described in the *Overview* chapter. If an unrecognized mnemonic is sent, a beep and a return code rejecting the key may be sent.

If NORESET has not been requested by **Set Session Parameters** (9), a reset function is performed before the keystroke string. This will reset all states that can be reset except input-inhibited states. Note that soft keyboard locks will also be reset.

The effect on the *teem* product emulation will be exactly as if the keystrokes had occurred at the keyboard. If the keyboard is locked then the bell will sound for each keystroke. If the emulation is busy then the keystrokes will be placed on the keyboard queue and processed when the emulator becomes free. Error code **4 (**host session busy) will be returned when the host is busy and the keyboard is locked.

SET CURSOR (40)

Sets the position of the cursor within the connected presentation space.

Prerequisite Calls

Connect Presentation Space (1)

Supplied Parameters

Function Number: Must be 40.

Data String: NA.

Data String Length: NA.

PS Position: Desired cursor position in the presentation space.

Returned Parameters

Return Code: The following codes are valid:

Code	Explanation
0	The function was successful.
1	Not currently connected to a presentation space.
7	Invalid presentation space position.
9	A system error occurred.

Remarks

A presentation space position is determined by the offset into the presentation space where position 1 is the top left corner of the screen display (row 1 and column 1), and the highest position number is the bottom right corner.

The IBM 5250 emulation supports a presentation space of 24 rows by 80 columns. When an error message is received from the host or when the operator presses the **SysReq** key, a 25th row is displayed. When row 25 is displayed, it is a valid area for this function.

SET MESSAGE TIMEOUT (134)

This function allows an application to change the timeout value used by *teemtalk* or *teem*X for internal messages. The default timeout value is 45 seconds.

Prerequisite Calls

Connect Presentation Space (1)

Supplied Parameters

Function Number: Must be 134.

Data String: NA.

Data String Length: NA.

PS Position: Message timeout value in seconds.
0 = no timeout.

Returned Parameters

Return Code: The following codes are valid:

Code	Explanation
0	The function was successful.
1	Not currently connected to a presentation space.
9	A system error occurred.

SET SESSION PARAMETERS (9)

This function enables you to change certain default global session parameters. The parameters remain in effect until changed by this function or a **Reset System** (21) is issued.

Prerequisite Calls

None.

Supplied Parameters

Function Number: Must be 9.

Data String: Contains the session options that are to be changed (see the following pages), separated by commas or spaces.

Data String Length: Exact length of the data string in bytes (STREOT is not allowed).

PS Position: NA.

Returned Parameters

Data String Length: For return code 2, number of valid parameters set, else unchanged.

Return Code: The following codes are valid:

Code	Explanation
0	The function was successful.
2	One or more parameters invalid.
9	A system error occurred.

Remarks

All correct parameters will be interpreted even if invalid parameters are present in the data string.

Session Options

The valid session options are as follows.

Data String Length

The following options specify how the data string length is determined.

STRLEN Default. An exact length is passed for all strings.

STREOT String lengths are not specifically encoded. They are terminated with an EOT character.

End Of Transmission Character

The following option specifies the end of transmission (EOT) character used to terminate a string when STREOT is set.

EOT=c "c" is the binary value of the EOT character, the default being binary zero. No spaces are to be included on either side of the equals sign. The space character is not a valid EOT character.

Extent Of Presentation Space Search

The following options specify how the presentation space is to be searched.

SRCHALL Default. A **Search Presentation Space** (6) call will search the entire presentation space, and **Search Field** (30) will search the entire field.

SRCHFROM If the session option SRCHFRWD is set, the search will start at the specified PS position and stop at the end of the field or presentation space. If SRCHBKWD is set, the search will start at the end of the presentation space or field and stop at the specified PS position.

Direction Of Search

The following options specify the direction in which a search is performed when the session option SRCHFROM is set.

SRCHFRWD Default. The search starts at the specified PS position and stops at the end of the field or presentation space.

SRCHBKWD The search starts at the specified PS position and stops at the specified PS position.

Pause Type

The following options determine the type of pause to use for the **Pause** (18) function.

FPAUSE Default. Pauses for the full length of time specified by the **Pause** (18) function.

IPAUSE Any host event can interrupt the pause once a **Start Host Notification** (23) call is made.

Reset Prior To Keystroke

The following options determine whether strings sent by the **Send Keystring** (3) function are automatically preceded with a reset.

AUTORESET Default. All strings sent by **Send Keystring** (3) are preceded with a reset.

NORESET Strings sent by **Send Keystring** (3) are not preceded with a reset.

Escape Character For Keystroke Mnemonics

The following option affects the **Send Keystring** (3) and **Get Key** (51) functions and specifies the literal character to be used to represent the escape character in keystroke mnemonics.

ESC=@ Specifies that **@** represents the escape character (default). The character must immediately follow the equals sign, without a space.

Keyboard Mapping

The following options affect the **Send Keystring** (3) and **Get Key** (51) functions and determine whether standard IBM or direct keyboard mapping is supported. The *ASCII Mnemonics* section in chapter 2 lists the functions supported by each option.

KEYSIBM Default. Enable standard IBM mapping.

KEYSKBD Enable direct keyboard mapping.

Trace Facility

The following options enable or disable the trace facility that is used to debug the API program.

TROFF Default. Turns trace off.

TRON Turns trace on. All API functions are traced to 'stderr', which can be piped to a file if required.

Wait Period Characteristics

The following options affect the **Wait** (4) and **Get Key** (51) functions, but in different ways.

TWAIT For **Wait** (4), waits up to one minute before timing out on keyboard lock (default).

For **Get Key** (51), control is not returned to the API program until it has intercepted a normal or AID key as specified by the **Start Keystroke Intercept** (50) function.

LWAIT For **Wait** (4), waits forever on keyboard lock.

For **Get Key** (51), control is not returned to the API program until it has intercepted a normal or AID key as specified by the **Start Keystroke Intercept** (50) function.

NWAIT For **Wait** (4), checks status and return immediately without waiting.

For **Get Key** (51), returns return code 25 (keystrokes not available) in the fourth parameter if the keystroke intercept queue does not contain anything that matches the option specified by the **Start Keystroke Intercept** (50) function.

Attribute Byte Treatment

The following options determine how attribute bytes are treated.

NOATTRB Default. All bytes that do not have ASCII equivalents will be converted to spaces.

ATTRB All bytes that do not have ASCII equivalents will be passed as their original values.

Extended Attribute Byte Return

The following options determine whether **Copy Presentation Space** (5) and **Copy Presentation Space To String** (8) return extended attribute bytes (EABs).

NOEAB Default. Pass data only.

EAB Pass presentation space data with extended attribute bytes. Two bytes will be returned for each position in the presentation space. The first byte is the data character, the second is the extended attribute byte. The EAB returned is determined by the NOXLATE/XLATE setting, as described in the next section.

Extended Attribute Byte Translation

The following options determine the type of extended attribute byte (EAB) returned by **Copy Presentation Space** (5) and **Copy Presentation Space To String** (8).

Note that in all modes except IBM 3270, IBM 5250 and ICL 7561, EABs are always returned as 0 (zero).

IBM 3270 EABs

NOXLATE Default. Standard EABs returned.

Bits	Meaning		
7 - 6	Visual attribute:	Hex 40	Blink
		80	Reverse video
		C0	Underline
5 - 3	Foreground colour:	0 Black	4 Green
		1 Blue	5 Turquoise
		2 Red	6 Yellow
		3 Pink	7 White
2 - 0	Reserved.		

XLATE EABs returned as PC CGA colours. The top four bits return the background colour, the bottom four bits return the foreground colour. The colours are as follows:

0 Black	6 Brown	12 Light red
1 Blue	7 White	13 Light magenta
2 Green	8 Grey	14 Yellow
3 Cyan	9 Light blue	15 High intensity white
4 Red	10 Light green	
5 Magenta	11 Light cyan	

IBM 5250 EABs

NOXLATE Default. Standard EABs returned.

Bits	Meaning
7	Reverse
6	Underscore
5	Blink
4	Column separators
3 - 0	Reserved

XLATE EABs returned as PC CGA colours. The top four bits return the background colour, the bottom four bits return the foreground colour. The colours are as follows:

0	Black	6	Brown	12	Light red
1	Blue	7	White	13	Light magenta
2	Green	8	Grey	14	Yellow
3	Cyan	9	Light blue	15	High intensity white
4	Red	10	Light green		
5	Magenta	11	Light cyan		

ICL 7561 EABs

Bits	Meaning
7	Highlight
6	Protected
5	Field attribute
4	Blank
3	Bold
2	Underline
1	Reverse video
0	Blink

Session To Use For A Connection

The following options specify whether a new or existing session is used for a connection.

USEANY Default. Use an existing *teemtalk* / *teem*X session, or a new session if one does not exist.

USENEW Use a new *teemtalk* / *teem*X session.

USEOLD Use an existing *teemtalk* / *teem*X session.

Non-Display (Hidden) Fields Return

The following options specify how non-display fields will be copied using **Copy Presentation Space** (5), **Copy Presentation Space To String** (8) or **Copy Field To String** (34).

DISPLAY Characters copied as for display fields.

NONDISPLAY Characters copied as spaces.

SET TEEMTALK / TEEMX PARAMETER (109)

Allows the setting of a *teemtalk* or *teem*X parameter.

Prerequisite Calls

Connect Presentation Space (1)

Supplied Parameters

Function Number: Must be 109.

Data String: String containing the *teemtalk* / *teem*X settings definition, as defined in the script language PSET command (see the *teem* product Programmer's Guide).

Data String Length: Length of the data string (in bytes). This parameter is not applicable if in end of transmission (EOT) mode (see **Set Session Parameters** (9)).

PS Position: NA.

Returned Parameters

Return Code: The following codes are valid:

Code	Explanation
0	The function was successful.
1	Not currently connected to a presentation space.
2	Invalid parameter(s)
9	A system error occurred.

Remarks

The valid *teemtalk* or *teem*X settings are those listed for the script language PSET command as documented in the *teem* product Programmer's Guide.

START HOST NOTIFICATION (23)

Begins the process by which **Pause** (18) and **Query Host Update** (24) can determine if the presentation space and/or status line (OIA) has been updated.

Prerequisite Calls

Connect Presentation Space (1)

Supplied Parameters

Function Number: Must be 23.

Data String: Contains the following data:

Offset	Definition
0	One character ID for presentation space, or blank for current presentation space.
1	**P** = for presentation space update only. **O** = for OIA update only. **B** = for presentation space and OIA updates.

Data String Length: NA (2 implied).

PS Position: NA.

Returned Parameters

Return Code: The following codes are valid:

Code	Explanation
0	The function was successful.
1	Not currently connected to a presentation space.
2	Invalid parameter.
9	A system error occurred.

Remarks

Currently only supports reporting for both presentation space and OIA updates.

START KEYSTROKE INTERCEPT (50)

This function will intercept keystokes sent to a session by a terminal operator and store them until the keystroke queue overflows or the **Stop Keystroke Intercept** (53) or **Reset System** (21) functions are called. The intercepted keystokes can be:

- received through **Get Key** (51) and sent to the current or another session by **Send Keystring** (3).
- substituted with other keystrokes by **Send Keystring** (3).
- accepted or rejected by **Post Intercept Status** (52).
- used to initiate a process.

Prerequisite Calls

Connect Presentation Space (1)

Supplied Parameters

Function Number: Must be 50.

Data String: Contains the following data:

Offset	Definition
0	One character ID for presentation space, or blank for current presentation space.
1	An **option code** character **D** = for AID keystrokes only **L** = for all keystrokes.
2 - 5	The data at these positions is ignored.

Data String Length: The number of bytes in the keystroke intercept buffer, 256 are recommended. API allocates a minimum of 8 bytes for this buffer.

PS Position: NA.

Returned Parameters

Return Code: The following codes are valid:

Code	Explanation
0	The function was successful.
1	Not currently connected to a presentation space.
2	Invalid parameter.
4	Function inhibited as target presentation space was busy.
9	A system error occurred. **Release** (12) is being used.

Remarks

If **Get Key** (51) returns code **31**, either increase the size of the keystroke intercept buffer for this function, or execute **Get Key** (51) more frequently to retrieve keystrokes from the buffer so that space is made available. Note that each intercepted keystroke occupies three bytes in the buffer.

If option code D is provided, intercepted non-AID keys will be written to the intended presentation space, and only AID keys will be returned to the application.

Your API program must call the **Stop Keystroke Intercept** (53) function before exiting, otherwise this function will remain active and produce unexpected results.

STOP HOST NOTIFICATION (25)

Stops the process by which **Pause** (18) and **Query Host Update** (24) can determine if the presentation space and/or status line (OIA) has been updated.

Prerequisite Calls

Start Host Notification (23)

Supplied Parameters

Function Number: Must be 25.

Data String: One character ID for presentation space, or blank for current presentation space.

Data String Length: NA (1 implied).

PS Position: NA.

Returned Parameters

Return Code: The following codes are valid:

Code	Explanation
0	The function was successful.
1	Invalid presentation space specified.
8	No prior **Start Host Notification** (23).
9	A system error occurred.

STOP KEYSTROKE INTERCEPT (53)

Stops the keystroke intercept process previously initiated by the **Start Keystroke Intercept** (50) function.

Prerequisite Calls

Start Keystroke Intercept (50)

Supplied Parameters

Function Number: Must be 53.

Data String: One character ID for presentation space, or blank for current presentation space.

Data String Length: NA (1 implied).

PS Position: NA.

Returned Parameters

Return Code: The following codes are valid:

Code	Explanation
0	The function was successful.
1	Invalid presentation space specified.
8	No prior **Start Keystroke Intercept** (50).
9	A system error occurred.

WAIT (4)

This function checks the host session for a wait condition. If the session is busy or the keyboard is locked, the API application will wait for the amount of time specified by the **Set Session Parameters** (9) function to see if the wait condition clears.

Prerequisite Calls

Connect Presentation Space (1)

Supplied Parameters

Function Number: Must be 4.

Data String: NA.

Data String Length: NA.

PS Position: NA.

Returned Parameters

Return Code: The following codes are valid:

Code	Explanation
0	The function was successful; the keyboard was unlocked and the emulator was not busy.
1	Not currently connected to a presentation space.
4	Time-out while still busy.
5	Keyboard locked.
9	A system error occurred.

Remarks

The **Wait** function can be used to allow functions such as **Send Keystring** (3) sufficient time to complete.

The behaviour of the **Wait** function is determined by the TWAIT, LWAIT or NWAIT setting specified by the **Set Session Parameters** (9) function.

If TWAIT is specified, the function waits for up to one minute for the emulator to become not busy and/or for an input-inhibited state to be cleared. If the emulator is busy and input is inhibited the function returns error code **4** (time-out while still busy).

If LWAIT is specified, the function waits indefinitely for the emulator to become not busy and for any input-inhibit state to be cleared. This option is not recommended as control does not return to the API application until the host is available.

If NWAIT is specified, the function returns immediately after checking the status of the *teem* product. Note that the return code **4** (time-out while still busy) is returned when the emulator is busy and in an input inhibit state.

Note that a successful return code **0** from a **Wait** function only indicates that the emulator is now free (not busy and unlocked) and not that a specific application has completed updating the screen. Use **Search Field** (30) or **Search Presentation Space** (6) combined with the **Wait** function to look for expected keyword prompts from the user application in order to determine the completion of a host update.

The 'busy' indication (on this and other functions) will be reported on ALL emulations while data is waiting to be processed from the host. Additionally, the 'busy' indication will be reported on the IBM 3270 emulation while XSYSTEM or XCLOCK is present, the IBM 5250 emulation while in a 'locked' state (awaiting host reply), and the HP700/92 emulation while in a 'transmit pending' state.

Wait is useful in waiting for the target computer to complete updating the presentation space after sending a command. The procedure relies on waiting for the presentation space to become busy indicating that the response is being received from the target computer. The API application should then wait for the presentation space to become not busy and finally search for an application specific prompt string or the presence of the cursor in a given location. The procedure should be repeated until the application specific prompt string is found in the presentation space or field specified.

The use of **Wait** in the above procedure reduces the processor load when waiting for the presentation space to become free.

Notes

A

Field Attribute Representation

This appendix describes how field attributes are represented in the data string returned by a query field attribute function.

Introduction

You can use one of three functions to provide a copy of the field attribute at the specified presentation space position:

Query Field Attribute (14)
Copy Formatted Presentation Space (105)
Query Formatted Field Attribute (114)

The following sections describe the meaning of the bytes returned by these functions.

Query Field Attribute

The information returned by the **Query Field Attribute** (14) function depends on the current terminal emulation.

VT Edit Mode:	Hex	Attribute
	01	Blink
	02	Reverse video
	04	Underline
	08	Bold
	10	Blank
	40	Mask for erasable character

HP700/92 Mode:	Hex	Attribute
	01	Blink
	02	Reverse video
	04	Underline
	08	Half intensity
	10	Security
	20	Line drawing
	40	Unprotected
	80	Protected

PT250 Mode:	Hex	Attribute
	07	Device type mask
	08	Modified data
	10	Must fill
	20	Must enter
	40	Selected area

TA6530 Mode:	Hex	Attribute
	01	Modified data
	0E	Data type mask
	10	Auto tab disable
	20	Protected
	40	Upshift
	80	Field attribute

Wyse Modes:	Hex	Attribute
	01	Blink
	02	Reverse video
	04	Underline
	08	Half intensity
	10	Blank
	20	Erasable
	40	Field attribute

IBM 3270 Mode:	Hex	Attribute
	01	Modified data
	0C	Intensity mask:
		00 Display
		04 Display, intense
		08 Selectable
		0C Hidden
	10	Numeric
	20	Protected
	C0	Field attribute

IBM 3151 Mode:	**Hex**	**Attribute**
	01	Blink
	02	Reverse video
	04	Underline
	08	Bold
	10	Blank
	20	Visible rendition
	40	Protected
	80	Modified data

IBM 5250 Mode:	**Hex**	**Attribute**
	01	Modified data
	0E	Field type:
		00 Alphanumeric
		02 Alphabetic
		04 Numeric shift
		06 Numeric only
		08 Reserved
		0A Digits only
		0C Magnetic stripe reader data only
		0E Signed numeric
	10	High intensity
	20	Protected
	40	Displayed
	80	Field attribute

ICL 7561 Mode:	**Hex**	**Attribute**
	01	Blink
	02	Reverse video
	04	Underline
	08	Bold
	10	Blank
	20	Field attribute
	40	Protected
	80	Highlight attribute set

Formatted Field & Presentation Space

The **Query Formatted Field Attribute** (114) and **Copy Formatted Presentation Space** (105) functions are used to provide a comprehensive description of the field attribute at the specified position in the presentation space. The returned data string consists of four bytes which provide the following information:

Byte 1: Character displayed (or Viewdata special character).
Byte 2: Foreground and background colour index.
Byte 3: Generally indicates the display attribute (underline, flashing, etc.).
Byte 4: Generally indicates field formatting.

The following sections list the values applicable for each byte and their meaning.

Byte 1: Character Displayed

The first byte generally indicates the character displayed at the specified PS position. Most emulations use the ISO Latin-1 character set, exceptions are as follows:

HP700/92 Emulation:	HP Roman 8 character set
IBM 3270 Emulation:	EBCDIC character set
IBM 5250 Emulation:	EBCDIC character set
Viewdata Mode:	ASCII character set

Viewdata mode supports the following special characters in addition to the ASCII set:

Hex	Meaning	Hex	Meaning
01	Begin Alpha	18	Conceal
07	End Alpha	19	Contiguous Graphics
08	Begin Flashing	1A	Separated Graphics
09	Begin Steady	1C	Black
11	Begin Graphic	1D	New
17	End Graphic	1E	Hold
0C	Normal	1F	Release
0D	Double		

Byte 2: Colour Indices

The second byte indicates the background and foreground colours for the specified PS position. The top four bits (hex nibble) specify the background colour and the bottom four bits (hex nibble) specify the foreground colour. For example, a hex value of 1B specifies a blue background and bold green foreground.

The following hex values apply to all terminal emulations except IBM 3270.

Hex	Colour	Hex	Colour
0	Black	8	Bold Black
1	Blue	9	Bold Blue
2	Red	A	Bold Red
3	Green	B	Bold Green
4	Mauve	C	Bold Mauve
5	Cyan	D	Bold Cyan
6	Yellow	E	Bold Yellow
7	White	F	Bold White

The following hex values apply to the IBM 3270 terminal emulation.

Hex	Colour	Hex	Colour
0	Neutral 0	8	Black
1	Blue	9	Deep Blue
4	Green	C	Pale Green
5	Turquoise	D	Pale Turquoise
6	Yellow	E	Grey
7	Neutral 7	F	White

Byte 3: Display Attribute

The third byte generally indicates the display attribute (underline, flashing, etc.). The meaning of the byte returned depends on the terminal emulation.

VT, DG410, AT&T4410 & Stratus V102 Modes:

Hex	Display Attribute
01	Blink
02	Reverse video
04	Underline
08	Bold
10	Blank
40	Mask for erasable characters

HP700/92 Mode:

Hex	Display Attribute
01	Blink
02	Reverse video
04	Underline
08	Half intensity
10	Security
20	Line drawing
40	Unprotected
80	Protected

PT250 Mode:	**Hex**	**Display Attribute**
	01	Half intensity
	02	Strike-thru
	04	Invisible image
	08	Blink
	10	Reverse video
	20	Underline
	C0	Character set mask

TA6530 Mode:	**Hex**	**Display Attribute**
	01	Half intensity
	02	Blink
	04	Reverse video
	08	Blank
	10	Underline
	20	Visual attribute start
	40	Field start

Wyse Modes:	**Hex**	**Display Attribute**
	01	Blink
	02	Reverse video
	04	Underline
	08	Half intensity
	10	Blank
	20	Erasable
	40	Field start

IBM 3270 Mode:	**Hex**	**Display Attribute**
	01	Blink
	02	Reverse video
	04	Underline
	07	Highlight mask

IBM 3151 Mode:	**Hex**	**Display Attribute**
	01	Blink
	02	Reverse video
	04	Underline
	08	Bold
	10	Blank
	20	Visible rendition
	40	Protected
	80	Modified data

IBM 5250 Mode:	**Hex**	**Display Attribute**
	01	Reverse video
	02	High intensity

	04	Underline
	08	Blink
	10	Column separator
	20	Visual attribute start
	40	Extended character set
	80	Input position

ICL 7561 Mode:	**Hex**	**Display Attribute**
	01	Blink
	02	Reverse video
	04	Underline
	08	Bold
	10	Blank
	20	Field attribute
	40	Protected
	80	Highlight attribute set

Byte 4: Field Formatting

The fourth byte generally indicates field formatting. The meaning of the byte returned depends on the terminal emulation.

VT, DG410, AT&T4410 & Stratus V102 Modes:

	Hex	**Display Attribute**
	07	Character font selection mask
	08	Double width
	10	Double height top
	20	Double height bottom
	80	Copy/cut/paste attribute

HP700/92 Mode:	**Hex**	**Field Formatting**
	01	Active row
	02	Start font
	04	Start protected/unprotected field
	08	Start video attribute field
	10	Start security field
	20	International font
	80	Copy/cut/paste attribute

PT250 Mode:	**Hex**	**Field Formatting**
	07	Device type mask
	08	Modified data
	10	Must fill
	20	Must enter

	40	Selected area
	80	Copy/cut/paste attribute

TA6530 Mode:	**Hex**	**Field Formatting**
	01	Device type mask
	0E	Data type mask
	10	Auto tab disable
	20	Protected field
	40	Upshift
	80	Copy/cut/paste attribute

Wyse Modes:	**Hex**	**Field Formatting**
	80	Copy/cut/paste attribute

IBM 3270 Mode:	**Hex**	**Field Formatting**
	01	Modified data
	0C	Display/select/intensity mask
	10	Numeric field
	20	Protected field
	40	Field start
	80	Copy/cut/paste attribute

IBM 3151 Mode:	**Hex**	**Field Formatting**
	08	Field attribute
	10	Numeric field
	20	Default field attribute
	80	Copy/cut/paste attribute

IBM 5250 Mode:	**Hex**	**Display Attribute**
	07	Field type mask
	08	Modified data
	10	DUP enable
	20	Bypass field
	40	Field start
	80	Copy/cut/paste attribute

ICL 7561 Mode:	**Hex**	**Display Attribute**
	01	Font
	02	Word wrap
	04	Modified field
	08	Numeric validation
	10	Alpha validation
	20	Space validation
	40	Auto tab attribute
	80	Copy/cut/paste attribute

B

teemtalk API Program Examples

This appendix shows example API programs for use with *teemtalk*.

Introduction

The following sections show example API files that are included with the Programmer's API Development Kit. The first section shows the files used to generate the **HAPITEST.EXE** (Windows) or **TT4W32HAPI.EXE** (Windows NT) program which enables you to perform any of the API functions simply by selecting from a list in a dialog box. The second section shows example files used to generate an API link between *teemtalk* and a Visual Basic application.

API Function Test Program

The Programmer's API Development Kit includes an executable file called **HAPITEST.EXE** (Windows) or **APITST32.EXE** (Windows NT) that enables you to perform any of the API functions simply by selecting from a list in a dialog box. Once installed, clicking the **HAPITEST** or **TT4W32HAPI** icon will display a window with a menu bar. Selecting **Invoke HLLAPI Function...** from the **Options** menu will display a dialog box giving you various options for actioning an API function.

The files used to generate the **HAPITEST.EXE** (Windows) or **APITST32.EXE** (Windows NT) program are also supplied as examples of how to write an API program. These files are listed below and their contents are shown on the following pages.

HAPIFUNC.H	Defines the functions for *teemtalk* (and *teem*X) API programs.

HAPITEST.C	Main routines for invoking API functions when running **HAPITEST.EXE** (Windows) or **APITST32.EXE** (Windows NT).
HAPITEST.H	Definitions used by **HAPITEST.C**, **HAPITEST.RC** and **HAPITEST.DLG**.
MAKEFILE	Generates the **HAPITEST.EXE** (Windows) or **APITST32.EXE** (Windows NT) program.
HAPITEST.DEF	Definitions used for linking.
HAPITEST.RC	Windows resources used by **HAPITEST.EXE** (Windows) or **APITST32.EXE** (Windows NT).
HAPITEST.DLG	Windows dialog boxes used by **HAPITEST.EXE** (Windows) or **APITST32.EXE** (Windows NT).

HAPIFUNC.H

```
/*********************************************************************\
*                                                                     *
*                                                                     *
*            Copyright (C) 1989-1996 Pericom Software PLC             *
*                         All Rights Reserved                         *
*                                                                     *
*                       Pericom Software PLC.                         *
*                            The Priory                               *
*                             Cosgrove                                *
*                           Milton Keynes                             *
*                          Bucks, MK19 7JJ                            *
*                              England                                *
*                                                                     *
\*********************************************************************/


/**************************/
/* HLLAPI Function Numbers */
/**************************/

#define HA_CONNECT_PS                1
#define HA_DISCONNECT_PS             2
#define HA_SEND_KEYSTRING            3
#define HA_WAIT                      4
#define HA_COPY_PS                   5
#define HA_SEARCH_PS                 6
#define HA_QUERY_CURSOR_LOC          7
#define HA_COPY_PS_TO_STRING         8
#define HA_SET_SESSION_PARAMS        9
#define HA_QUERY_SESSIONS           10
#define HA_RESERVE                  11
#define HA_RELEASE                  12
#define HA_COPY_OIA                 13
#define HA_QUERY_FIELD_ATTR         14
#define HA_COPY_STRING_TO_PS        15
/* not used 16 */
#define HA_STORAGE_MANAGER          17
#define HA_PAUSE                    18
/* not used 19 */
#define HA_QUERY_SYSTEM             20
#define HA_RESET_SYSTEM             21
#define HA_QUERY_SESSION_STATUS     22
#define HA_START_HOST_NOTIF         23
#define HA_QUERY_HOST_UPDATE        24
#define HA_STOP_HOST_NOTIF          25
/* not used 26 */
/* not used 27 */
/* not used 28 */
/* not used 29 */
#define HA_SEARCH_FIELD             30
#define HA_FIND_FIELD_POSITION      31
#define HA_FIND_FIELD_LENGTH        32
```

```
#define HA_COPY_STRING_TO_FIELD      33
#define HA_COPY_FIELD_TO_STRING      34
/* not used 35 */
/* not used 36 */
/* not used 37 */
/* not used 38 */
/* not used 39 */
#define HA_SET_CURSOR                40
/* not used 41 */
/* not used 42 */
/* not used 43 */
/* not used 44 */
/* not used 45 */
/* not used 46 */
/* not used 47 */
/* not used 48 */
/* not used 49 */
#define HA_START_KEY_INTERCEPT       50
#define HA_GETKEY                    51
#define HA_POST_INTERCEPT_STAT       52
#define HA_STOP_KEY_INTERCEPT        53
/* not used 54 to 89 inclusive */
#define HA_SEND_FILE                 90
#define HA_RECEIVE_FILE              91
/* not used 92 */
/* not used 93 */
/* not used 94 */
/* not used 95 */
/* not used 96 */
/* not used 97 */
/* not used 98 */
#define HA_CONVERT_POSITION          99
#define HA_CONVERT_ROWCOL            99
/* not used 100 */
/* not used 101 */
/* not used 102 */
#define HA_SEND_KEYPRESS            103
/* not used 104 */
#define HA_COPY_FORM_PS             105
/* not used 106 */
/* not usd 107 */
#define HA_COPY_FORM_PS_TO_STRING   108
#define HA_SET_TEEMTALK_PARAM       109
#define HA_GET_TEEMTALK_PARAM       110
/* not used 111 */
/* not used 112 */
/* not used 113 */
#define HA_QUERY_FORM_FIELD_ATTR    114
/* not used 115 */
/* not used 116 */
/* not used 117 */
/* not used 118 */
/* not used 119 */
/* not used 120 */
/* not used 121 */
```

```
/* not used 122 */
/* not used 123 */
/* not used 124 */
/* not used 125 */
/* not used 126 */
/* not used 127 */
#define HA_PERICOM_FUNCTIONS        128
#define HA_HOST_INPUT               129
#define HA_WINDOW_HANDLE            130


/************************************************/
/* HLLAPI Pericom Functions (128) Function Numbers */
/************************************************/
#define HA_PERIFUNC_SAVESET     1


/***********************/
/* HLLAPI Return Values */
/***********************/

#define HA_STATUS_CODE_INVALID_PS   9998
#define HA_STATUS_CODE_PARAM_ERROR  9999

#define HARC_SUCCESS                 0
#define HARC_NO_HOST_UPDATES         0
#define HARC_INVALID_PS              1
#define HARC_PARAM_ERROR             2
/* not used 3 */
#define HARC_BUSY_PS                 4
#define HARC_FUNCTION_INHIBITED      5
#define HARC_ILLEGAL_DATA            5
#define HARC_DATA_ERROR              6
#define HARC_INVALID_PS_POSITION     7
#define HARC_NO_PREREQ_ISSUED        8
#define HARC_SYSTEM_ERROR            9
#define HARC_FUNC_NOT_SUPPORTED     10
#define HARC_RESOURCE_UNAVAIL       11
/* not used 12 */
/* not used 13 */
/* not used 14 */
/* not used 15 */
/* not used 16 */
/* not used 17 */
/* not used 18 */
/* not used 19 */
#define HARC_INVALID_KEYSTROKE      20
#define HARC_OIA_UPDATED            21
#define HARC_PS_UPDATED             22
#define HARC_OIA_AND_PS_UPDATED     23
#define HARC_NOT_FOUND              24
#define HARC_PS_UNFORMATTED         24
#define HARC_KEYS_NOT_AVAIL         25
#define HARC_HOST_EVENT_OCCURED     26
/* not used 27 */
```

```
#define HARC_FIXED_LENGTH_ZERO      28
/* not used 29 */
/* not used 30 */
#define HARC_KEY_QUEUE_OVERFLOW     31


/*****************************/
/* HLLAPI Extern declarations */
/*****************************/
#ifdef __cplusplus
extern "C" {            /* Assume C declarations for C++ */
#endif   /* __cplusplus */
#ifdef WINVER
#ifdef WIN32
__declspec(dllexport)
WORD    WINAPI  hllapi(
                    LPWORD  lpwFuncNum,
                    LPBYTE  lpDataString,
                    LPWORD  lpwDataStringLength,
                    LPWORD  lpwRetCode
                    );
__declspec(dllexport)
int     WINAPI  open_workstation(
                    int     iShortname,
                    LPSTR   lpArgs
                    );
__declspec(dllexport)
int     WINAPI  close_workstation(
                    int     iShortname
                    );
#else
int FAR PASCAL  hllapi(
                    LPINT   lpiFuncNum,
                    LPSTR   lpDataString,
                    LPINT   lpiDataStringLength,
                    LPINT   lpiRetCode
                    );
int     WINAPI  open_workstation(
                    int     iShortname,
                    LPSTR   lpArgs
                    );
int     WINAPI  close_workstation(
                    int     iShortname
                    );
#endif
#else
int             hllc(
                 /* int     *function_number,       */
                 /* char    *data_string,           */
                 /* int     *data_string_length,    */
                 /* int     *return_code            */
                    );
int             open_workstation(
                 /* int     iShortname,             */
                 /* char    *lpArgs                 */
```

```
                    );
int             close_workstation(
                 /* int     iShortname              */
                    );
#endif
#ifdef __cplusplus
}                       /* End of extern “C” { */
#endif   /* __cplusplus */
```

HAPITEST.C

```
/*********************************************************************\
*                                                                     *
*                                                                     *
*            Copyright (C) 1989-1996 Pericom Software PLC             *
*                         All Rights Reserved                         *
*                                                                     *
*                       Pericom Software PLC.                         *
*                            The Priory                               *
*                             Cosgrove                                *
*                           Milton Keynes                             *
*                          Bucks, MK19 7JJ                            *
*                              England                                *
*                                                                     *
\*********************************************************************/


/*************************************************************************

    PROGRAM: HAPITEST.C

    PURPOSE: Main Windows I/F routines for FTP for Windows

    FUNCTIONS:
     WinMain() - calls initialization function, processes message loop
     InitApplication() - initializes window data and registers window
     InitInstance() - saves instance handle and creates main window
     MainWndProc() - processes messages
        FuncDlgProc() - processes messages for “Function Call” dialog box
        AboutDlgProc() - processes messages for “About” dialog box
        ShutDownDlgProc() - processes messages for “ShutDown” dialog box

    COMMENTS:
        Windows can have several copies of your application running at the
        same time.  The variable hInst keeps track of which instance this
        application is so that processing will be to the correct window.

*************************************************************************/

#include <windows.h>                /* required for all Windows applications */
#if defined WIN32
#include <windowsx.h>
#endif
#include <string.h>
#include <stdlib.h>
#include <ctype.h>

#include “hapifunc.h”               /* specific to this program            */
#include “hapitest.h”


HANDLE  hInst;                      /* current instance                    */
BOOL    bUserAbort  = FALSE;
```

```
HWND    hWndMain    = (HWND)NULL;   /* Main window handle.                */

HANDLE      hScrnRows;
int FAR     *ScrnRows;
HANDLE      hDisplayBuffer  = (HANDLE)NULL;
BYTE FAR    *DisplayBuffer;
int         iMaxVLines      = 0;
int         SRowPosn        = 0;
int         DBuffPosn       = 0;
int         nVscrollPos     = 0;

char        szAppName[]     = "HllapiTest";


/*************************************************************************

    FUNCTION: WinMain(HANDLE, HANDLE, LPSTR, int)

    PURPOSE: calls initialization function, processes message loop

    COMMENTS:

        Windows recognizes this function by name as the initial entry point
        for the program.  This function calls the application initialization
        routine, if no other instance of the program is running, and always
        calls the instance initialization routine.  It then executes a message
        retrieval and dispatch loop that is the top-level control structure
        for the remainder of execution.  The loop is terminated when a WM_QUIT
        message is received, at which time this function exits the application
        instance by returning the value passed by PostQuitMessage().

        If this function must abort before entering the message loop, it
        returns the conventional value NULL.

*************************************************************************/

int PASCAL WinMain(hInstance, hPrevInstance, lpCmdLine, nCmdShow)
HANDLE  hInstance;                      /* current instance             */
HANDLE  hPrevInstance;                  /* previous instance            */
LPSTR   lpCmdLine;                      /* command line                 */
int     nCmdShow;                       /* show-window type (open/icon) */
{
MSG     msg;                            /* message                      */

    if (!hPrevInstance)                 /* Other instances of app running? */
        if (!InitApplication(hInstance))    /* Initialize shared things */
            return (FALSE);             /* Exits if unable to initialize     */

    /* Perform initializations that apply to a specific instance */

    if (!InitInstance(hInstance, nCmdShow))
        return (FALSE);

    /* Acquire and dispatch messages until a WM_QUIT message is received. */
```

```
    while (GetMessage(&msg,         /* message structure                      */
            NULL,                   /* handle of window receiving the message */
            0,                      /* lowest message to examine              */
            0))                     /* highest message to examine             */
    {
        TranslateMessage(&msg);     /* Translates virtual key codes           */
        DispatchMessage(&msg);      /* Dispatches message to window           */
    }
    return (msg.wParam);            /* Returns the value from PostQuitMessage */
}


/****************************************************************************

    FUNCTION: InitApplication(HANDLE)

    PURPOSE: Initializes window data and registers window class

    COMMENTS:

        This function is called at initialization time only if no other
        instances of the application are running.  This function performs
        initialization tasks that can be done once for any number of running
        instances.

        In this case, we initialize a window class by filling out a data
        structure of type WNDCLASS and calling the Windows RegisterClass()
        function.  Since all instances of this application use the same window
        class, we only need to do this when the first instance is initialized.


****************************************************************************/

BOOL
InitApplication(hInstance)
HANDLE      hInstance;                  /* current instance           */
{
WNDCLASS    wc;

    /* Fill in window class structure with parameters that describe the       */
    /* main window.                                                           */

    wc.style = CS_HREDRAW | CS_VREDRAW;
    wc.lpfnWndProc = MainWndProc;       /* Function to retrieve messages for  */
                                        /* windows of this class.             */
    wc.cbClsExtra = 0;                  /* No per-class extra data.           */
    wc.cbWndExtra = 0;                  /* No per-window extra data.          */
    wc.hInstance = hInstance;           /* Application that owns the class.   */
    wc.hIcon = LoadIcon(hInstance, “HAPIICON”);
    wc.hCursor = LoadCursor(NULL, IDC_ARROW);
    wc.hbrBackground = GetStockObject(WHITE_BRUSH);
    wc.lpszMenuName =  “HapiTestMenu”;  /* Name of menu resource in .RC file. */
    wc.lpszClassName = “HapiTestClass”; /* Name used in call to CreateWindow. */

    /* Register the window class and return success/failure code. */
```

```
    return (RegisterClass(&wc));
}


/****************************************************************************

    FUNCTION:  InitInstance(HANDLE, int)

    PURPOSE:  Saves instance handle and creates main window

    COMMENTS:

        This function is called at initialization time for every instance of
        this application.  This function performs initialization tasks that
        cannot be shared by multiple instances.

        In this case, we save the instance handle in a static variable and
        create and display the main program window.

****************************************************************************/

BOOL InitInstance(hInstance, nCmdShow)
HANDLE  hInstance;                      /* Current instance identifier.       */
int     nCmdShow;                       /* Param for first ShowWindow() call. */
{
HWND    hWnd;                           /* Main window handle.                */

    /* Save the instance handle in static variable, which will be used in  */
    /* many subsequence calls from this application to Windows.            */

    hInst = hInstance;

    /* Create a main window for this application instance.  */

    hWnd = CreateWindow(
        "HapiTestClass",                /* See RegisterClass() call.          */
        szAppName,                      /* Text for window title bar.         */
                                        /* Window style.                      */
        WS_OVERLAPPEDWINDOW | WS_VSCROLL | WS_HSCROLL,
        CW_USEDEFAULT,                  /* Default horizontal position.       */
        CW_USEDEFAULT,                  /* Default vertical position.         */
        CW_USEDEFAULT,                  /* Default width.                     */
        CW_USEDEFAULT,                  /* Default height.                    */
        NULL,                           /* Overlapped windows have no parent. */
        NULL,                           /* Use the window class menu.         */
        hInstance,                      /* This instance owns this window.    */
        NULL                            /* Pointer not needed.                */
        );

    /* If window could not be created, return "failure" */

    if (!hWnd)
        return (FALSE);
    else
        hWndMain = hWnd;
```

```
    /* Make the window visible; update its client area; and return “success” */

    ShowWindow(hWnd, nCmdShow); /* Show the window                        */
    UpdateWindow(hWnd);         /* Sends WM_PAINT message                 */
    return (TRUE);              /* Returns the value from PostQuitMessage */

}

/****************************************************************************

    FUNCTION: MainWndProc(HWND, unsigned, WORD, LONG)

    PURPOSE:  Processes messages

    MESSAGES:

     WM_COMMAND    - application menu (About dialog box)
     WM_DESTROY    - destroy window

    COMMENTS:

     To process the IDM_ABOUT message, call MakeProcInstance() to get the
     current instance address of the About() function.  Then call Dialog
     box which will create the box according to the information in your
     generic.rc file and turn control over to the About() function.  When
     it returns, free the intance address.

****************************************************************************/

long WINAPI
MainWndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
#if !defined WIN32
FARPROC         lpDlgProc;
static  long    cxChar,
                cyChar,
                cxClient,
                cyClient;
#else
static  int     cxChar,
                cyChar,
                cxClient,
                cyClient;
#endif
static  char    Strn1[16];
static  char    Strn2[16];
static  char    InpBuffer[128];
static  int     nVscrollMax,
                nHscrollPos,
                nHscrollMax;
HDC             hdc;
short           n,
                y,
                nVscrollInc,
                nHscrollInc,
```

```
                OldVSize;
PAINTSTRUCT     ps;
TEXTMETRIC      tm;
BYTE       FAR *lpOutText;
int             iTempDispRow;
LOGFONT         lf;
static HFONT    hFont,
                hOldFont;
BOOL            bAction;

    switch (message)
    {
   case WM_CREATE:
        hdc = GetDC(hWnd);

        GetObject(GetStockObject(SYSTEM_FONT), sizeof(LOGFONT), (LPSTR)&lf);
        lf.lfPitchAndFamily = FIXED_PITCH | FF_ROMAN;
        lf.lfWeight = FW_NORMAL;

        hFont = CreateFontIndirect(&lf);
        hOldFont = SelectObject(hdc, hFont);

        GetTextMetrics (hdc, &tm) ;
        cxChar = tm.tmAveCharWidth ;
        cyChar = tm.tmHeight + tm.tmExternalLeading ;

        SelectObject(hdc, hOldFont);

        ReleaseDC (hWnd, hdc) ;
                                                    /* Allocate Display Buffer */
        hDisplayBuffer = GlobalAlloc(GMEM_FIXED | GMEM_ZEROINIT, (MAXNUMLINES *
MAXDISPWIDTH));
        DisplayBuffer = (BYTE FAR *)GlobalLock(hDisplayBuffer);
                                  /* Allocate  & initialise screen row buffer */
        hScrnRows = GlobalAlloc(GMEM_MOVEABLE | GMEM_ZEROINIT, sizeof(int) *
MAXNUMLINES);
        ScrnRows = (int FAR *)GlobalLock(hScrnRows);

        for (n = 0; n < MAXNUMLINES; n++)
            ScrnRows[n] = n;

        return 0 ;

   case WM_SIZE:
        cxClient = LOWORD(lParam);
        cyClient = HIWORD(lParam);

        OldVSize = iMaxVLines;

        iMaxVLines = cyClient / cyChar;
        nVscrollMax = max (0, (MAXNUMLINES - iMaxVLines));

        SetScrollRange (hWnd, SB_VERT, 0, nVscrollMax, FALSE) ;
        SetScrollPos   (hWnd, SB_VERT, nVscrollPos, TRUE) ;
```

```
        nHscrollMax = max (0, ((MAXDISPWIDTH) - (cxClient / cxChar)));
        nHscrollPos = min (nHscrollPos, nHscrollMax) ;

        SetScrollRange (hWnd, SB_HORZ, 0, nHscrollMax, FALSE) ;
        SetScrollPos   (hWnd, SB_HORZ, nHscrollPos, TRUE) ;

        GlobalUnlock(hScrnRows);
        hScrnRows = GlobalReAlloc(hScrnRows, sizeof(int) * iMaxVLines, GMEM_MOVEABLE |
GMEM_ZEROINIT);
        ScrnRows = (int FAR *)GlobalLock(hScrnRows);
                    /* If screen has been increased, provide for more lines */
        if ((OldVSize < iMaxVLines) && OldVSize != 0)
        {
            iTempDispRow =  ScrnRows[OldVSize - 1] + 1;
            for (n = OldVSize; n < iMaxVLines; n++)
                ScrnRows[n] = iTempDispRow++;
        }

        return 0 ;

   case WM_VSCROLL:
        switch (wParam)
        {
       case SB_TOP:
            nVscrollInc = -nVscrollPos ;
            break ;

       case SB_BOTTOM:
            nVscrollInc = nVscrollMax - nVscrollPos ;
            break ;

       case SB_LINEUP:
            nVscrollInc = -1 ;
            break ;

       case SB_LINEDOWN:
            nVscrollInc = 1 ;
            break ;

       case SB_PAGEUP:
            nVscrollInc = min (-1, -cyClient / cyChar) ;
            break ;

       case SB_PAGEDOWN:
            nVscrollInc = max(1, cyClient / cyChar) ;
            break ;

       case SB_THUMBTRACK:
            nVscrollInc = LOWORD(lParam) - nVscrollPos ;
            break ;

       default:
            nVscrollInc = 0 ;
        }
```

```
         nVscrollInc = max (-nVscrollPos,
         min (nVscrollInc, nVscrollMax - nVscrollPos)) ;

         if (nVscrollInc != 0)
         {
             nVscrollPos += nVscrollInc ;
//             ScrollWindow (hWnd, 0, -cyChar * nVscrollInc, NULL, NULL) ;
             SetScrollPos (hWnd, SB_VERT, nVscrollPos, TRUE) ;
             ScrollScreenRows(nVscrollInc);
             InvalidateRect(hWndMain, NULL, TRUE);
         }
         return 0 ;

    case WM_HSCROLL:
         switch (wParam)
         {
         case SB_LINEUP:
              nHscrollInc = -1 ;
              break ;

         case SB_LINEDOWN:
              nHscrollInc = 1 ;
              break ;

         case SB_PAGEUP:
              nHscrollInc = -8 ;
              break ;

         case SB_PAGEDOWN:
              nHscrollInc = 8 ;
              break ;

         case SB_THUMBPOSITION:
              nHscrollInc = LOWORD (lParam) - nHscrollPos ;
              break ;

         default:
              nHscrollInc = 0 ;
         }

         nHscrollInc = max (-nHscrollPos, min (nHscrollInc, nHscrollMax - nHscrollPos))
;

         if (nHscrollInc != 0)
         {
             nHscrollPos += nHscrollInc ;
//             ScrollWindow (hWnd, -cxChar * nHscrollInc, 0, NULL, NULL) ;
             SetScrollPos (hWnd, SB_HORZ, nHscrollPos, TRUE) ;
             InvalidateRect(hWndMain, NULL, TRUE);
         }
         return 0 ;

    case WM_PAINT:
         hdc = BeginPaint (hWnd, &ps) ;
         hOldFont = SelectObject(hdc, hFont);
```

```
        for (n = 0; n < iMaxVLines; n++)
        {
            y = cyChar * n;

            lpOutText = &DisplayBuffer[(ScrnRows[n] * MAXDISPWIDTH) + nHscrollPos];
            TextOut(hdc, 0, y, lpOutText, lstrlen(lpOutText));
        }

        SelectObject(hdc, hOldFont);

        EndPaint (hWnd, &ps) ;
        return 0 ;


   case WM_COMMAND:           /* message: command from application menu */
        switch(GET_WM_COMMAND_ID(wParam, lParam))
        {
       case IDM_ABOUT:
#if !defined WIN32
            lpDlgProc = MakeProcInstance(AboutDlgProc, hInst);

            DialogBox(hInst,"AboutBox", hWnd, lpDlgProc);
            FreeProcInstance(lpDlgProc);
#else
            DialogBox(hInst,"AboutBox", hWnd, AboutDlgProc);
#endif
            break;

       case IDM_TEST_EXIT:
#if !defined WIN32
            lpDlgProc = MakeProcInstance(ShutDownDlgProc, hInst);
            DialogBox(hInst,"ShutDown", hWndMain, lpDlgProc);
            FreeProcInstance(lpDlgProc);
#else
            DialogBox(hInst,"AboutBox", hWnd, ShutDownDlgProc);
#endif
            if (bUserAbort)
                PostMessage(hWndMain,WM_DESTROY,0,0L);
            break;

       case IDM_FUNCTION:
#if !defined WIN32
            lpDlgProc = MakeProcInstance(FuncDlgProc, hInst);
            bAction = TRUE;
            while (bAction)
                bAction = DialogBox(hInst, "FunctionBox", hWnd, lpDlgProc);
            FreeProcInstance(lpDlgProc);
#else
            bAction = TRUE;
            while (bAction)
         bAction = DialogBox(hInst, "FunctionBox", hWnd, FuncDlgProc);
#endif
            break;
```

```
        case IDM_CLEAR:
             MEMSET(DisplayBuffer, 0, MAXDISPWIDTH * MAXNUMLINES);

             for (n = 0; n < iMaxVLines; n++)
                 ScrnRows[n] = n;

             nVscrollPos = 0;
             SetScrollPos (hWnd, SB_VERT, 0, TRUE);
             SRowPosn = 0;
             DBuffPosn = 0;

             InvalidateRect(hWndMain, NULL, TRUE);
             break;

        default:                                /* Lets Windows process it */
             return (DefWindowProc(hWnd, message, wParam, lParam));
         }
         break;

    case WM_DESTROY:                    /* message: window being destroyed */
         DeleteObject(hFont);
         GlobalUnlock(hScrnRows);
         GlobalFree(hScrnRows);
         GlobalUnlock(hDisplayBuffer);
         GlobalFree(hDisplayBuffer);
         PostQuitMessage(0);
         break;

    default:                            /* Passes it on if unproccessed    */
         return (DefWindowProc(hWnd, message, wParam, lParam));
     }

     return (0L);
}


/****************************************************************************

    FUNCTION: FuncDlgProc(HWND, unsigned, WORD, LONG)

    PURPOSE:  Processes messages for "Function" dialog box

    MESSAGES:

     WM_INITDIALOG - initialize dialog box
     WM_COMMAND    - Input received

****************************************************************************/
BOOL WINAPI
FuncDlgProc(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)
{
WORD            iFuncNum;
static  char    szDataString[1025];
WORD            iDataStrLen;
BOOL            bTranslated;
```

```
WORD            iPSPosition;
UINT            n,
                iIndex;

    switch (message)
    {
   case WM_INITDIALOG:
        for (n = HLLAPI_FUNCTION_BASE; n < IDT_UNKNOWN_FUNC; n++)
        {
            if (!LoadString(hInst, n, szDataString, 128))
                continue;

            iIndex = SendDlgItemMessage(hDlg, IDD_TEST_FUNCNUMBER, CB_ADDSTRING, 0,
(LPARAM)(LPSTR)szDataString);

            SendDlgItemMessage(hDlg, IDD_TEST_FUNCNUMBER, CB_SETITEMDATA, iIndex,
(LPARAM)n - HLLAPI_FUNCTION_BASE);
        }

        SendDlgItemMessage(hDlg, IDD_TEST_SESSID, CB_ADDSTRING, 0,
(LPARAM)(LPSTR)”None”);
        szDataString[1] = ‘\0’;
        for (n = ‘A’; n <= ‘Z’; n++)
        {
            szDataString[0] = (char)n;
            SendDlgItemMessage(hDlg, IDD_TEST_SESSID, CB_ADDSTRING, 0,
(LPARAM)(LPSTR)szDataString);
        }
        SendDlgItemMessage(hDlg, IDD_TEST_SESSID, CB_ADDSTRING, 0,
(LPARAM)(LPSTR)”Blank”);

        SendDlgItemMessage(hDlg, IDD_TEST_FUNCNUMBER, CB_SETCURSEL, 0, 0L);
        SendDlgItemMessage(hDlg, IDD_TEST_SESSID, CB_SETCURSEL, 0, 0L);
        SendDlgItemMessage(hDlg, IDD_TEST_DATASTR, EM_LIMITTEXT, 1023, 0L);
        SendDlgItemMessage(hDlg, IDD_TEST_STRLEN, EM_LIMITTEXT, 4, 0L);
        SendDlgItemMessage(hDlg, IDD_TEST_PSPOSN, EM_LIMITTEXT, 6, 0L);

        SetDlgItemInt(hDlg, IDD_TEST_STRLEN, 0, FALSE);
        SetDlgItemInt(hDlg, IDD_TEST_PSPOSN, 0, FALSE);

        return TRUE;

   case WM_COMMAND:
        if (GET_WM_COMMAND_ID(wParam, lParam) == IDOK)
        {
            iIndex   = SendDlgItemMessage(hDlg, IDD_TEST_FUNCNUMBER, CB_GETCURSEL, 0,
0L);
         iFuncNum = (WORD)SendDlgItemMessage(hDlg, IDD_TEST_FUNCNUMBER,
CB_GETITEMDATA, iIndex, 0L);

            iIndex   = SendDlgItemMessage(hDlg, IDD_TEST_SESSID, CB_GETCURSEL, 0, 0L);
            if (iIndex != 0 && iIndex <= ‘Z’)
            {
                szDataString[0] = iIndex - 1 + ‘A’;
                GetDlgItemText(hDlg, IDD_TEST_DATASTR, &szDataString[1], 1023);
```

```
            }
            else if (iIndex == 0)
                GetDlgItemText(hDlg, IDD_TEST_DATASTR, szDataString, 1023);
            else
            {
                szDataString[0] = ' ';
                GetDlgItemText(hDlg, IDD_TEST_DATASTR, &szDataString[1], 1023);
            }

            iDataStrLen = GetDlgItemInt(hDlg, IDD_TEST_STRLEN, &bTranslated, FALSE);
            if (bTranslated == 0)
            {
                MessageBox(hWndMain, "String Length MUST be a Number, not text.",
"Parameter Error", MB_OK);
                return TRUE;
            }

            iPSPosition = GetDlgItemInt(hDlg, IDD_TEST_PSPOSN, &bTranslated, FALSE);
            if (bTranslated == 0)
            {
                MessageBox(hWndMain, "PS Position MUST be a Number, not text.",
"Parameter Error", MB_OK);
                return TRUE;
            }

            EndDialog(hDlg, TRUE);
            InvokeFunction(iFuncNum, szDataString, iDataStrLen, iPSPosition);
        }
        else if (wParam == IDCANCEL)
            EndDialog(hDlg, FALSE);
        else if (wParam == IDD_TEST_STRCALC)
        {
            iIndex   = SendDlgItemMessage(hDlg, IDD_TEST_SESSID, CB_GETCURSEL, 0, 0L);
            if (iIndex != 0 && iIndex <= 'Z')
            {
                szDataString[0] = iIndex - 1 + 'A';
                GetDlgItemText(hDlg, IDD_TEST_DATASTR, &szDataString[1], 1023);
            }
            else if (iIndex == 0)
                GetDlgItemText(hDlg, IDD_TEST_DATASTR, szDataString, 1023);
            else
            {
                szDataString[0] = ' ';
                GetDlgItemText(hDlg, IDD_TEST_DATASTR, &szDataString[1], 1023);
            }

            iIndex = ParseString(szDataString, szDataString, 1023);
            SetDlgItemInt(hDlg, IDD_TEST_STRLEN, iIndex, FALSE);
        }

        return (TRUE);
    }

    return (FALSE);                             /* Didn't process a message */
}
```

```
/*****************************************************************************

    FUNCTION: AboutDlgProc(HWND, unsigned, WORD, LONG)

    PURPOSE:  Processes messages for "About" dialog box

    MESSAGES:

     WM_INITDIALOG - initialize dialog box
     WM_COMMAND    - Input received

    COMMENTS:

     No initialization is needed for this particular dialog box, but TRUE
     must be returned to Windows.

     Wait for user to click on "Ok" button, then close the dialog box.

*****************************************************************************/
BOOL WINAPI
AboutDlgProc(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)
{
char    szBuffer[128];

    switch (message)
    {
   case WM_INITDIALOG:
        wsprintf((LPSTR)szBuffer, (LPSTR)"Version %s", (LPSTR)IDS_VERSNUM_STR);
        SetDlgItemText(hDlg,IDD_TEST_VERSNUM,(LPSTR)szBuffer);

        LoadString(hInst, IDT_TEST_PROGNAME, szBuffer,127);
        SetDlgItemText(hDlg,IDD_TEST_PROGNAME,(LPSTR)szBuffer);

        return (TRUE);

   case WM_COMMAND:                    /* message: received a command  */
        if (GET_WM_COMMAND_ID(wParam, lParam) == IDOK)
        {                              /* "OK" box selected?           */
            EndDialog(hDlg, TRUE);     /* Exits the dialog box         */

            return (TRUE);
        }
    }

    return (FALSE);                            /* Didn't process a message */
}



/*****************************************************************************

    FUNCTION: ShutDownDlgProc(HWND, unsigned, WORD, LONG)

    PURPOSE:  Processes messages for "Shutdown" dialog box
```

```
*************************************************************************/

BOOL WINAPI
ShutDownDlgProc(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)
{
    switch (message)
    {
   case WM_INITDIALOG:
        SetWindowText(hDlg,"Shut Down Dialogue");

        SetDlgItemText(hDlg,IDD_TEST_SHUTDOWN,"Do you really want to shutdown?");

        SetDlgItemText(hDlg,IDD_TEST_PROGNAME,szAppName);

        SetDlgItemText(hDlg,IDOK,"OK");

        SetDlgItemText(hDlg,IDCANCEL,"Cancel");

     bUserAbort = FALSE;
     break;

   case WM_COMMAND:
        switch (GET_WM_COMMAND_ID(wParam, lParam))
        {
       case IDOK:
            bUserAbort = TRUE;
       case IDCANCEL:
            EndDialog(hDlg, TRUE);
            break;

       default:
            return FALSE;
        }
        break;

    default:
        return(FALSE);
     }

    return (TRUE);
}


void
InvokeFunction(WORD iFuncNum, LPSTR szDataString, WORD iDataStrLen, WORD iPSPosition)
{
HANDLE           hData;
WORD            iDataLen,
                OutLogAvail;
LPSTR           szData;
static  char    lpAPIDataString[32768];
WORD            FunctionNumber,
                DataStringLen,
                len,
```

```
                PresPosn;
int             iResFunc;
int             iRetCode;
char            szBuffer[MAXDISPWIDTH + 1];
char            FunctionName[48];
char            ErrorString[48];
int             iProcessID;
                                        /* Ensure can read Control Codes */
    iDataLen = lstrlen(szDataString);
    hData = GlobalAlloc(GMEM_FIXED | GMEM_ZEROINIT, iDataLen + 1);
    szData = GlobalLock(hData);
    iDataLen = ParseString(szData, szDataString, iDataLen);

                                        /* Create buffer to pass to HLLAPI */
    MEMSET(lpAPIDataString, '\0',   (UINT)32768);
    if (iDataLen)
        MEMCPY(lpAPIDataString, szData, iDataLen);

    FunctionNumber = iFuncNum;
    DataStringLen  = iDataStrLen;
    PresPosn       = iPSPosition;

    iResFunc = (FunctionNumber + HLLAPI_FUNCTION_BASE);
    if (!GetNameFromFuncNum(FunctionNumber, (LPSTR)FunctionName, &OutLogAvail))
    {
        wsprintf(szBuffer, "Invalid Function Number %d", FunctionNumber);
        OutputTextString(szBuffer);
    }
    else
    if (iResFunc >= IDT_WS_START)
    {
        wsprintf(szBuffer, "Calling Function %d (%s)", FunctionNumber,
(LPSTR)FunctionName);
        OutputTextString(szBuffer);

        wsprintf(szBuffer, "        Session  %c", *lpAPIDataString);
        OutputTextString(szBuffer);

        if (iResFunc == IDT_WS_OPEN)
        {
            wsprintf(szBuffer, "        Args     %s", (LPSTR)&lpAPIDataString[1]);
            OutputTextString(szBuffer);
        }

        switch (iResFunc)
        {
       case IDT_WS_OPEN:
            iRetCode = open_workstation((int)*lpAPIDataString,
(LPSTR)&lpAPIDataString[1]);
            break;

       case IDT_WS_CLOSE:
            iRetCode = close_workstation((int)*lpAPIDataString);
            break;
```

```
        default:
             iRetCode = (int)(0 - HARC_SYSTEM_ERROR);
             break;
         }

         if (iRetCode >= 0)
         {
             iProcessID = iRetCode;
             iRetCode = 0;
         }
         else
         {
             iProcessID = 0;
             iRetCode = -iRetCode;
         }

         GetErrorFromErnum(iRetCode, (LPSTR)ErrorString);

         wsprintf(szBuffer, “got Return Code  %d (%s)”, iRetCode, (LPSTR)ErrorString);
         OutputTextString(szBuffer);

         if ((iResFunc == IDT_WS_OPEN) && (iProcessID))
         {
             wsprintf(szBuffer, “        Process  %04X”, iProcessID);
             OutputTextString(szBuffer);
         }
    }
    else
    {
         wsprintf(szBuffer, “Calling Function %d (%s)”, FunctionNumber,
(LPSTR)FunctionName);
         OutputTextString(szBuffer);

         wsprintf(szBuffer, “        Length   %d”, DataStringLen);
         OutputTextString(szBuffer);

         wsprintf(szBuffer, “        PS posn  %d”, PresPosn);
         OutputTextString(szBuffer);

         switch (FunctionNumber)
         {
         case HA_CONNECT_PS              :
             len = 1;
             break;

         case HA_DISCONNECT_PS           :
         case HA_WAIT                    :
         case HA_COPY_PS                 :
         case HA_QUERY_CURSOR_LOC        :
         case HA_COPY_PS_TO_STRING       :
         case HA_QUERY_SESSIONS          :
         case HA_RESERVE                 :
         case HA_RELEASE                 :
         case HA_COPY_OIA                :
         case HA_QUERY_FIELD_ATTR        :
```

```
case HA_STORAGE_MANAGER         :
case HA_QUERY_SYSTEM            :
case HA_RESET_SYSTEM            :
case HA_STOP_HOST_NOTIF         :
case HA_COPY_FIELD_TO_STRING    :
case HA_SET_CURSOR              :
case HA_CONVERT_POSITION        :
case HA_COPY_FORM_PS            :
case HA_COPY_FORM_PS_TO_STRING  :
case HA_QUERY_FORM_FIELD_ATTR   :
case HA_WINDOW_HANDLE           :
    len = 0;
    break;

case HA_PAUSE                   :
case HA_QUERY_SESSION_STATUS    :
case HA_QUERY_HOST_UPDATE       :
case HA_GETKEY                  :
case HA_STOP_KEY_INTERCEPT      :
    len = 1;
    break;

case HA_START_HOST_NOTIF        :
case HA_FIND_FIELD_POSITION     :
case HA_FIND_FIELD_LENGTH       :
case HA_START_KEY_INTERCEPT     :
case HA_POST_INTERCEPT_STAT     :
    len = 2;
    break;

case HA_SET_SESSION_PARAMS      :
    len = DataStringLen;
    break;

case HA_PERICOM_FUNCTIONS       :
    switch (PresPosn)
    {
    case HA_PERIFUNC_SAVESET    :
        len = 0;
        break;

    default :
        len = DataStringLen;
        break;
    }
    break;

case HA_SEND_KEYSTRING          :
case HA_SEARCH_PS               :
case HA_COPY_STRING_TO_PS       :
case HA_SEARCH_FIELD            :
case HA_COPY_STRING_TO_FIELD    :
case HA_SEND_FILE               :
case HA_RECEIVE_FILE            :
case HA_SEND_KEYPRESS           :
```

```
        case HA_SET_TEEMTALK_PARAM      :
        case HA_GET_TEEMTALK_PARAM      :
        case HA_HOST_INPUT              :
        default :
            len = DataStringLen;
            break;
        }
        OutputFormatData(lpAPIDataString, len);

        iRetCode = hllapi(&FunctionNumber, lpAPIDataString, &DataStringLen,
&PresPosn);

        GetErrorFromErnum(iRetCode, (LPSTR)ErrorString);

        wsprintf(szBuffer, “got Return Code  %d (%s)”, iRetCode, (LPSTR)ErrorString);
        OutputTextString(szBuffer);

        wsprintf(szBuffer, “        Length   %d”, DataStringLen);
        OutputTextString(szBuffer);

        wsprintf(szBuffer, “        PS posn  %d”, PresPosn);
        OutputTextString(szBuffer);

        if (iRetCode == 0)
        {
            if (OutLogAvail)
                OutputFormatData(lpAPIDataString, DataStringLen);
            else
            {
                switch (FunctionNumber)
                {
               case HA_QUERY_SESSIONS :
                    OutputFormatData(lpAPIDataString, (WORD)(DataStringLen * 12));
                    break;

               case HA_GETKEY :
                    OutputFormatData(lpAPIDataString, 8);
                    break;
                }
            }
        }
    }
    MEMSET(szBuffer, ‘*’, MAXDISPWIDTH - 1);
    szBuffer[MAXDISPWIDTH] = ‘\0’;
    OutputTextString(szBuffer);

    GlobalUnlock(hData);
    GlobalFree(hData);
}



/* Parse String, converting controls & _nnn */
/*————————————————————*/
```

```
int
ParseString(unsigned char  far *dst,unsigned char  far *src,int  count)
{
int     n,
        i;
char    digits[4];

    i = 0;
    n = 0;
    for(n = 0; n < count; )
    {
        if(src[i] == '\0')
            break;

        if(src[i] == '^')
        {
            if(src[i+1] >= '@' && src[i+1] <= '_') /* control code, e.g ^B */
                dst[n++] = src[i+1] - '@';
            else
            {
                dst[n++] = src[i];
                if(src[i+1] == '\0')
                    break;
                dst[n++] = src[i+1];
            }
            i += 2;
        }

        else if(src[i] == '\\')
        {
            if(src[i+1] == '^')         /* \^ sequence gives ^ */
                dst[n++] = src[i+1];
            else if(src[i+1] == '\0')
            {
                dst[n++] = src[i];
                break;
            }
            else
            {
                dst[n++] = src[i];
                dst[n++] = src[i+1];
            }
            i += 2;
        }
        else if(src[i] == '_' && isdigit(src[i+1]) && isdigit(src[i+2]) &&
isdigit(src[i+3]))
        {
            digits[0] = src[i+1];       /* _027 gives decimal 27 (ESC) */
            digits[1] = src[i+2];
            digits[2] = src[i+3];
            digits[3] = '\0';
            dst[n++] = (char)atoi(digits);
            i += 4;
        }
        else
```

```
            dst[n++] = src[i++];
    }
    dst[n] = '\0';
    return(n);
}

/*****************************************************************************

    FUNCTION: OutputText(BYTE FAR *lpOutString)

    PURPOSE:  Displays Output string resulting from API calls.

*****************************************************************************/


void
OutputTextString(BYTE FAR * lpOutString)
{
int     n,
        iOffSet,
        iRowLimit;
int     cbOutString;

    iRowLimit = iMaxVLines - 1;

    if (DBuffPosn == BUFFERLIMIT)       /* Full buffers, effect full scroll */
    {
        /* Make sure write @ E.O. Display buffer ie cater for poss scroll */
        if (ScrnRows[iRowLimit] != DBuffPosn)
        {
            iOffSet = DBuffPosn;

            for (n = iRowLimit; n >= 0; n—)
               ScrnRows[n] = iOffSet—;
        }

        for (n = 0, iOffSet = 0; n < BUFFERRANGE; n++, iOffSet += MAXDISPWIDTH)
            lstrcpyn(&DisplayBuffer[n], &DisplayBuffer[n + MAXDISPWIDTH],
MAXDISPWIDTH);
        cbOutString = min(lstrlen(lpOutString), (MAXDISPWIDTH - 1));
        lstrcpyn(&DisplayBuffer[DBuffPosn * MAXDISPWIDTH], lpOutString, (cbOutString +
1));

        nVscrollPos =  DBuffPosn - iRowLimit;
        SetScrollPos(hWndMain, SB_VERT, nVscrollPos, TRUE);
    }
    else
    {
        /* If not yet filled screen rows */
        if (SRowPosn < iRowLimit)
        {
            iOffSet = DBuffPosn;
            for (n = SRowPosn; n >= 0; n—)
                ScrnRows[n] = iOffSet—;
            SRowPosn++;
```

```
        }
        else                            /* Not @ E.O. DisplayBuffer */
        {                               /* But filled available screen */
            iOffSet = DBuffPosn;
            for (n = iRowLimit; n >= 0; n—)
                ScrnRows[n] = iOffSet—;

            nVscrollPos =  DBuffPosn - iRowLimit;
            SetScrollPos(hWndMain, SB_VERT, nVscrollPos, TRUE);
        }

        cbOutString = min(lstrlen(lpOutString), (MAXDISPWIDTH - 1));
        lstrcpyn(&DisplayBuffer[(DBuffPosn++) * MAXDISPWIDTH], lpOutString,
(cbOutString + 1));
    }

    InvalidateRect(hWndMain, NULL, TRUE);
}


void
ScrollScreenRows(int nVscrollInc)
{
int     n;
int     InitDispBuffer;

    InitDispBuffer = ScrnRows[iMaxVLines - 1] + nVscrollInc;

    for (n = (iMaxVLines - 1); n >= 0; n—)
        ScrnRows[n] = InitDispBuffer—;
}


int
GetNameFromFuncNum(WORD FuncNumber, LPSTR FuncName, WORD *OutLogAvail)
{
int     iRetCode    = TRUE;
int     iResFunc;

    *OutLogAvail = FALSE;

    iResFunc = FuncNumber + HLLAPI_FUNCTION_BASE;
    switch(FuncNumber)
    {
   case HA_CONNECT_PS:
   case HA_DISCONNECT_PS:
   case HA_SEND_KEYSTRING:
   case HA_WAIT:
   case HA_SEARCH_PS:
   case HA_QUERY_CURSOR_LOC:
   case HA_SET_SESSION_PARAMS:
   case HA_QUERY_SESSIONS:
   case HA_RESERVE:
   case HA_RELEASE:
   case HA_QUERY_FIELD_ATTR:
```

```
    case HA_COPY_STRING_TO_PS:
    case HA_PAUSE:
    case HA_RESET_SYSTEM:
    case HA_START_HOST_NOTIF:
    case HA_QUERY_HOST_UPDATE:
    case HA_STOP_HOST_NOTIF:
    case HA_SEARCH_FIELD:
    case HA_FIND_FIELD_POSITION:
    case HA_FIND_FIELD_LENGTH:
    case HA_COPY_STRING_TO_FIELD:
    case HA_SET_CURSOR:
    case HA_START_KEY_INTERCEPT:
    case HA_GETKEY:
    case HA_POST_INTERCEPT_STAT:
    case HA_STOP_KEY_INTERCEPT:
    case HA_CONVERT_POSITION:
    case HA_SEND_KEYPRESS:
    case HA_SET_TEEMTALK_PARAM:
    case HA_PERICOM_FUNCTIONS:
    case HA_HOST_INPUT :
         break;

    case HA_COPY_PS:
    case HA_COPY_PS_TO_STRING:
    case HA_COPY_OIA:
    case HA_QUERY_SYSTEM:
    case HA_QUERY_SESSION_STATUS:
    case HA_COPY_FIELD_TO_STRING:
    case HA_COPY_FORM_PS:
    case HA_COPY_FORM_PS_TO_STRING:
    case HA_QUERY_FORM_FIELD_ATTR:
    case HA_GET_TEEMTALK_PARAM:
    case HA_WINDOW_HANDLE:
         *OutLogAvail = TRUE;
         break;

    default:
         if ((iResFunc >= IDT_WS_START) && (iResFunc <= IDT_WS_END))
             break;
         iRetCode = FALSE;
         iResFunc = IDT_UNKNOWN_FUNC;
         break;
     }

     LoadString(hInst, iResFunc, FuncName, 60);

     return iRetCode;
}

void
GetErrorFromErnum(int ErrorNumber, LPSTR ErrorName)
{
int     iResError;

     switch(ErrorNumber)
```

```
     {
    case HARC_SUCCESS             :  /*  0 */
    case HARC_INVALID_PS          :  /*  1 */
    case HARC_PARAM_ERROR         :  /*  2 */
    case HARC_BUSY_PS             :  /*  4 */
    case HARC_FUNCTION_INHIBITED  :  /*  5 */
    case HARC_DATA_ERROR          :  /*  6 */
    case HARC_INVALID_PS_POSITION :  /*  7 */
    case HARC_NO_PREREQ_ISSUED    :  /*  8 */
    case HARC_SYSTEM_ERROR        :  /*  9 */
    case HARC_FUNC_NOT_SUPPORTED  :  /* 10 */
    case HARC_RESOURCE_UNAVAIL    :  /* 11 */
    case HARC_INVALID_KEYSTROKE   :  /* 20 */
    case HARC_OIA_UPDATED         :  /* 21 */
    case HARC_PS_UPDATED          :  /* 22 */
    case HARC_OIA_AND_PS_UPDATED  :  /* 23 */
    case HARC_NOT_FOUND           :  /* 24 */
    case HARC_KEYS_NOT_AVAIL      :  /* 25 */
    case HARC_HOST_EVENT_OCCURED  :  /* 26 */
    case HARC_FIXED_LENGTH_ZERO   :  /* 28 */
    case HARC_KEY_QUEUE_OVERFLOW  :  /* 31 */
         iResError = HLLAPI_ERROR_BASE + ErrorNumber;
         break;

   default:
         iResError = IDT_UNKNOWN_ERR;
         break;
     }

     LoadString(hInst, iResError, ErrorName, 60);

     return;
}

void
OutputFormatData(LPCSTR lpAPIDataString, WORD DataStringLen)
{
static char     TraceFormat[]   = "    %4s  %47.47s  %s";
int             index;
unsigned char   ch;
char            Line[10];
char            Hex[50];
char            Char[20];
int             iHex;
int             iChar;
char            szBuffer[MAXDISPWIDTH + 1];

    wsprintf(szBuffer, "  Data String:-");
    OutputTextString(szBuffer);

    Line[0] = 0;

    for (index = 0; index < DataStringLen; index++)
    {
        if (!(index % 16))
```

```
        {
            if (Line[0])
            {
                wsprintf(szBuffer, TraceFormat, (LPSTR)Line, (LPSTR)Hex, (LPSTR)Char);
                OutputTextString(szBuffer);
            }

            wsprintf(Line, "%04X", index);
            MEMSET(Hex, ' ', 47);
            Hex[0] = 0;
            iHex = 0;
            Char[0] = 0;
            iChar = 0;
        }
        ch = lpAPIDataString[index];
        wsprintf(&Hex[iHex], "%02X ", ch);
        iHex += 3;
        Hex[iHex] = ' ';
        Char[iChar] = ((((ch & 0x7f) >= ' ') && ((ch & 0x7f) < 0x7f)) ? (ch & 0x7f) :
'.');
        Char[++iChar] = 0;
    }

    if (Line[0])
    {
        wsprintf(szBuffer, TraceFormat, (LPSTR)Line, (LPSTR)Hex, (LPSTR)Char);
        OutputTextString(szBuffer);
    }

    wsprintf(szBuffer, "  String Length     %d", DataStringLen);
    OutputTextString(szBuffer);
}
```

HAPITEST.H

```
#define IDM_TEST_EXIT           100
#define IDM_ABOUT               110
#define IDM_FUNCTION            120
#define IDM_CLEAR               130

#define IDM_MESSAGE             200

#define IID_TEST_FUNCTION       300
#define IDD_TEST_FUNCNUMBER     301
#define IDD_TEST_DATASTR        302
#define IDD_TEST_STRLEN         303
#define IDD_TEST_PSPOSN         304
#define IDD_TEST_SESSID         305
#define IDD_TEST_STRCALC        306

#define IDD_TEST_ICON           1000
#define IDD_TEST_PROGNAME       1100
#define IDD_TEST_SHUTDOWN       1200
#define IDD_TEST_VERSNUM        1300

#define IDT_TEST_PROGNAME       2000

#define MAXNUMLINES             100
#define MAXDISPWIDTH            80
#define BUFFERLIMIT             (MAXNUMLINES - 1)
#define BUFFERRANGE             (MAXDISPWIDTH * BUFFERLIMIT)

/* StringTable Defines. */
/************************/

/************************\
 *                      *
 * Function Values      *
 *                      *
\************************/
#define HLLAPI_FUNCTION_BASE    3000

#define IDT_CONNECT_PS              HLLAPI_FUNCTION_BASE + HA_CONNECT_PS
#define IDT_DISCONNECT_PS           HLLAPI_FUNCTION_BASE + HA_DISCONNECT_PS
#define IDT_SEND_KEYSTRING          HLLAPI_FUNCTION_BASE + HA_SEND_KEYSTRING
#define IDT_WAIT                    HLLAPI_FUNCTION_BASE + HA_WAIT
#define IDT_COPY_PS                 HLLAPI_FUNCTION_BASE + HA_COPY_PS
#define IDT_SEARCH_PS               HLLAPI_FUNCTION_BASE + HA_SEARCH_PS
#define IDT_QUERY_CURSOR_LOC        HLLAPI_FUNCTION_BASE + HA_QUERY_CURSOR_LOC
#define IDT_COPY_PS_TO_STRING       HLLAPI_FUNCTION_BASE + HA_COPY_PS_TO_STRING
#define IDT_SET_SESSION_PARAMS      HLLAPI_FUNCTION_BASE + HA_SET_SESSION_PARAMS
#define IDT_QUERY_SESSIONS          HLLAPI_FUNCTION_BASE + HA_QUERY_SESSIONS
#define IDT_RESERVE                 HLLAPI_FUNCTION_BASE + HA_RESERVE
#define IDT_RELEASE                 HLLAPI_FUNCTION_BASE + HA_RELEASE
#define IDT_COPY_OIA                HLLAPI_FUNCTION_BASE + HA_COPY_OIA
#define IDT_QUERY_FIELD_ATTR        HLLAPI_FUNCTION_BASE + HA_QUERY_FIELD_ATTR
#define IDT_COPY_STRING_TO_PS       HLLAPI_FUNCTION_BASE + HA_COPY_STRING_TO_PS
```

```
#define IDT_FUNC_16                 HLLAPI_FUNCTION_BASE + 16
#define IDT_FUNC_17                 HLLAPI_FUNCTION_BASE + 17
#define IDT_PAUSE                   HLLAPI_FUNCTION_BASE + HA_PAUSE
#define IDT_FUNC_19                 HLLAPI_FUNCTION_BASE + 19
#define IDT_QUERY_SYSTEM            HLLAPI_FUNCTION_BASE + HA_QUERY_SYSTEM
#define IDT_RESET_SYSTEM            HLLAPI_FUNCTION_BASE + HA_RESET_SYSTEM
#define IDT_QUERY_SESSION_STATUS    HLLAPI_FUNCTION_BASE + HA_QUERY_SESSION_STATUS
#define IDT_START_HOST_NOTIF        HLLAPI_FUNCTION_BASE + HA_START_HOST_NOTIF
#define IDT_QUERY_HOST_UPDATE       HLLAPI_FUNCTION_BASE + HA_QUERY_HOST_UPDATE
#define IDT_STOP_HOST_NOTIF         HLLAPI_FUNCTION_BASE + HA_STOP_HOST_NOTIF
#define IDT_FUNC_26                 HLLAPI_FUNCTION_BASE + 26
#define IDT_FUNC_27                 HLLAPI_FUNCTION_BASE + 27
#define IDT_FUNC_28                 HLLAPI_FUNCTION_BASE + 28
#define IDT_FUNC_29                 HLLAPI_FUNCTION_BASE + 29
#define IDT_SEARCH_FIELD            HLLAPI_FUNCTION_BASE + HA_SEARCH_FIELD
#define IDT_FIND_FIELD_POSITION     HLLAPI_FUNCTION_BASE + HA_FIND_FIELD_POSITION
#define IDT_FIND_FIELD_LENGTH       HLLAPI_FUNCTION_BASE + HA_FIND_FIELD_LENGTH
#define IDT_COPY_STRING_TO_FIELD    HLLAPI_FUNCTION_BASE + HA_COPY_STRING_TO_FIELD
#define IDT_COPY_FIELD_TO_STRING    HLLAPI_FUNCTION_BASE + HA_COPY_FIELD_TO_STRING
#define IDT_FUNC_35                 HLLAPI_FUNCTION_BASE + 35
#define IDT_FUNC_36                 HLLAPI_FUNCTION_BASE + 36
#define IDT_FUNC_37                 HLLAPI_FUNCTION_BASE + 37
#define IDT_FUNC_38                 HLLAPI_FUNCTION_BASE + 38
#define IDT_FUNC_39                 HLLAPI_FUNCTION_BASE + 39
#define IDT_SET_CURSOR              HLLAPI_FUNCTION_BASE + HA_SET_CURSOR
#define IDT_FUNC_41                 HLLAPI_FUNCTION_BASE + 41
#define IDT_FUNC_42                 HLLAPI_FUNCTION_BASE + 42
#define IDT_FUNC_43                 HLLAPI_FUNCTION_BASE + 43
#define IDT_FUNC_44                 HLLAPI_FUNCTION_BASE + 44
#define IDT_FUNC_45                 HLLAPI_FUNCTION_BASE + 45
#define IDT_FUNC_46                 HLLAPI_FUNCTION_BASE + 46
#define IDT_FUNC_47                 HLLAPI_FUNCTION_BASE + 47
#define IDT_FUNC_48                 HLLAPI_FUNCTION_BASE + 48
#define IDT_FUNC_49                 HLLAPI_FUNCTION_BASE + 49
#define IDT_START_KEY_INTERCEPT     HLLAPI_FUNCTION_BASE + HA_START_KEY_INTERCEPT
#define IDT_GETKEY                  HLLAPI_FUNCTION_BASE + HA_GETKEY
#define IDT_POST_INTERCEPT_STATUS   HLLAPI_FUNCTION_BASE + HA_POST_INTERCEPT_STAT
#define IDT_STOP_KEY_INTERCEPT      HLLAPI_FUNCTION_BASE + HA_STOP_KEY_INTERCEPT
#define IDT_FUNC_54                 HLLAPI_FUNCTION_BASE + 54
#define IDT_FUNC_55                 HLLAPI_FUNCTION_BASE + 55
#define IDT_FUNC_56                 HLLAPI_FUNCTION_BASE + 56
#define IDT_FUNC_57                 HLLAPI_FUNCTION_BASE + 57
#define IDT_FUNC_58                 HLLAPI_FUNCTION_BASE + 58
#define IDT_FUNC_59                 HLLAPI_FUNCTION_BASE + 59
#define IDT_FUNC_60                 HLLAPI_FUNCTION_BASE + 60
#define IDT_FUNC_61                 HLLAPI_FUNCTION_BASE + 61
#define IDT_FUNC_62                 HLLAPI_FUNCTION_BASE + 62
#define IDT_FUNC_63                 HLLAPI_FUNCTION_BASE + 63
#define IDT_FUNC_64                 HLLAPI_FUNCTION_BASE + 64
#define IDT_FUNC_65                 HLLAPI_FUNCTION_BASE + 65
#define IDT_FUNC_66                 HLLAPI_FUNCTION_BASE + 66
#define IDT_FUNC_67                 HLLAPI_FUNCTION_BASE + 67
#define IDT_FUNC_68                 HLLAPI_FUNCTION_BASE + 68
#define IDT_FUNC_69                 HLLAPI_FUNCTION_BASE + 69
#define IDT_FUNC_70                 HLLAPI_FUNCTION_BASE + 70
```

```
#define IDT_FUNC_71                 HLLAPI_FUNCTION_BASE + 71
#define IDT_FUNC_72                 HLLAPI_FUNCTION_BASE + 72
#define IDT_FUNC_73                 HLLAPI_FUNCTION_BASE + 73
#define IDT_FUNC_74                 HLLAPI_FUNCTION_BASE + 74
#define IDT_FUNC_75                 HLLAPI_FUNCTION_BASE + 75
#define IDT_FUNC_76                 HLLAPI_FUNCTION_BASE + 76
#define IDT_FUNC_77                 HLLAPI_FUNCTION_BASE + 77
#define IDT_FUNC_78                 HLLAPI_FUNCTION_BASE + 78
#define IDT_FUNC_79                 HLLAPI_FUNCTION_BASE + 79
#define IDT_FUNC_80                 HLLAPI_FUNCTION_BASE + 80
#define IDT_FUNC_81                 HLLAPI_FUNCTION_BASE + 81
#define IDT_FUNC_82                 HLLAPI_FUNCTION_BASE + 82
#define IDT_FUNC_83                 HLLAPI_FUNCTION_BASE + 83
#define IDT_FUNC_84                 HLLAPI_FUNCTION_BASE + 84
#define IDT_FUNC_85                 HLLAPI_FUNCTION_BASE + 85
#define IDT_FUNC_86                 HLLAPI_FUNCTION_BASE + 86
#define IDT_FUNC_87                 HLLAPI_FUNCTION_BASE + 87
#define IDT_FUNC_88                 HLLAPI_FUNCTION_BASE + 88
#define IDT_FUNC_89                 HLLAPI_FUNCTION_BASE + 89
#define IDT_FUNC_90                 HLLAPI_FUNCTION_BASE + 90
#define IDT_FUNC_91                 HLLAPI_FUNCTION_BASE + 91
#define IDT_FUNC_92                 HLLAPI_FUNCTION_BASE + 92
#define IDT_FUNC_93                 HLLAPI_FUNCTION_BASE + 93
#define IDT_FUNC_94                 HLLAPI_FUNCTION_BASE + 94
#define IDT_FUNC_95                 HLLAPI_FUNCTION_BASE + 95
#define IDT_FUNC_96                 HLLAPI_FUNCTION_BASE + 96
#define IDT_FUNC_97                 HLLAPI_FUNCTION_BASE + 97
#define IDT_FUNC_98                 HLLAPI_FUNCTION_BASE + 98
#define IDT_CONVERT_POSITION        HLLAPI_FUNCTION_BASE + HA_CONVERT_POSITION
#define IDT_FUNC_100                HLLAPI_FUNCTION_BASE + 100
#define IDT_FUNC_101                HLLAPI_FUNCTION_BASE + 101
#define IDT_FUNC_102                HLLAPI_FUNCTION_BASE + 102
#define IDT_SEND_KEYPRESS           HLLAPI_FUNCTION_BASE + HA_SEND_KEYPRESS
#define IDT_FUNC_104                HLLAPI_FUNCTION_BASE + 104
#define IDT_COPY_FORM_PS            HLLAPI_FUNCTION_BASE + HA_COPY_FORM_PS
#define IDT_FUNC_106                HLLAPI_FUNCTION_BASE + 106
#define IDT_FUNC_107                HLLAPI_FUNCTION_BASE + 107
#define IDT_COPY_FORM_PS_TO_STRING  HLLAPI_FUNCTION_BASE + HA_COPY_FORM_PS_TO_STRING
#define IDT_SET_TEEMTALK_PARAM      HLLAPI_FUNCTION_BASE + HA_SET_TEEMTALK_PARAM
#define IDT_GET_TEEMTALK_PARAM      HLLAPI_FUNCTION_BASE + HA_GET_TEEMTALK_PARAM
#define IDT_FUNC_111                HLLAPI_FUNCTION_BASE + 111
#define IDT_FUNC_112                HLLAPI_FUNCTION_BASE + 112
#define IDT_FUNC_113                HLLAPI_FUNCTION_BASE + 113
#define IDT_QUERY_FORM_FIELD_ATTR   HLLAPI_FUNCTION_BASE + HA_QUERY_FORM_FIELD_ATTR
#define IDT_FUNC_115                HLLAPI_FUNCTION_BASE + 115
#define IDT_FUNC_116                HLLAPI_FUNCTION_BASE + 116
#define IDT_FUNC_117                HLLAPI_FUNCTION_BASE + 117
#define IDT_FUNC_118                HLLAPI_FUNCTION_BASE + 118
#define IDT_FUNC_119                HLLAPI_FUNCTION_BASE + 119
#define IDT_FUNC_120                HLLAPI_FUNCTION_BASE + 120
#define IDT_FUNC_121                HLLAPI_FUNCTION_BASE + 121
#define IDT_FUNC_122                HLLAPI_FUNCTION_BASE + 122
#define IDT_FUNC_123                HLLAPI_FUNCTION_BASE + 123
#define IDT_FUNC_124                HLLAPI_FUNCTION_BASE + 124
#define IDT_FUNC_125                HLLAPI_FUNCTION_BASE + 125
```

```
#define IDT_FUNC_126                HLLAPI_FUNCTION_BASE + 126
#define IDT_FUNC_127                HLLAPI_FUNCTION_BASE + 127
#define IDT_PERICOM_FUNCTIONS       HLLAPI_FUNCTION_BASE + HA_PERICOM_FUNCTIONS
#define IDT_HOST_INPUT              HLLAPI_FUNCTION_BASE + HA_HOST_INPUT
#define IDT_WINDOW_HANDLE           HLLAPI_FUNCTION_BASE + HA_WINDOW_HANDLE

#define HLLAPI_FUNCTION_LIMIT       IDT_WINDOW_HANDLE + 1

#define IDT_WS_START                HLLAPI_FUNCTION_LIMIT
#define IDT_WS_OPEN                 HLLAPI_FUNCTION_LIMIT + 0
#define IDT_WS_CLOSE                HLLAPI_FUNCTION_LIMIT + 1
#define IDT_WS_END                  HLLAPI_FUNCTION_LIMIT + 1

#define IDT_UNKNOWN_FUNC            IDT_WS_END + 1



/***********************\
 *                     *
 * Error/Return Values  *
 *                     *
\***********************/
#define HLLAPI_ERROR_BASE       4000

#define IDT_HARC_SUCCESS                (HLLAPI_ERROR_BASE + HARC_SUCCESS
)
#define IDT_HARC_INVALID_PS             (HLLAPI_ERROR_BASE + HARC_INVALID_PS
)
#define IDT_HARC_PARAM_ERROR            (HLLAPI_ERROR_BASE + HARC_PARAM_ERROR
)
#define IDT_HARC_BUSY_PS                (HLLAPI_ERROR_BASE + HARC_BUSY_PS
)
#define IDT_HARC_FUNCTION_INHIBITED     (HLLAPI_ERROR_BASE + HARC_FUNCTION_INHIBITED
)
#define IDT_HARC_DATA_ERROR             (HLLAPI_ERROR_BASE + HARC_DATA_ERROR
)
#define IDT_HARC_INVALID_PS_POSITION    (HLLAPI_ERROR_BASE + HARC_INVALID_PS_POSITION
)
#define IDT_HARC_NO_PREREQ_ISSUED       (HLLAPI_ERROR_BASE + HARC_NO_PREREQ_ISSUED
)
#define IDT_HARC_SYSTEM_ERROR           (HLLAPI_ERROR_BASE + HARC_SYSTEM_ERROR
)
#define IDT_HARC_FUNC_NOT_SUPPORTED     (HLLAPI_ERROR_BASE + HARC_FUNC_NOT_SUPPORTED
)
#define IDT_HARC_RESOURCE_UNAVAIL       (HLLAPI_ERROR_BASE + HARC_RESOURCE_UNAVAIL
)
#define IDT_HARC_INVALID_KEYSTROKE      (HLLAPI_ERROR_BASE + HARC_INVALID_KEYSTROKE
)
#define IDT_HARC_OIA_UPDATED            (HLLAPI_ERROR_BASE + HARC_OIA_UPDATED
)
#define IDT_HARC_PS_UPDATED             (HLLAPI_ERROR_BASE + HARC_PS_UPDATED
)
#define IDT_HARC_OIA_AND_PS_UPDATED     (HLLAPI_ERROR_BASE + HARC_OIA_AND_PS_UPDATED
)
#define IDT_HARC_NOT_FOUND              (HLLAPI_ERROR_BASE + HARC_NOT_FOUND
```

```
)
#define IDT_HARC_KEYS_NOT_AVAIL         (HLLAPI_ERROR_BASE + HARC_KEYS_NOT_AVAIL
)
#define IDT_HARC_HOST_EVENT_OCCURED     (HLLAPI_ERROR_BASE + HARC_HOST_EVENT_OCCURED
)
#define IDT_HARC_FIXED_LENGTH_ZERO      (HLLAPI_ERROR_BASE + HARC_FIXED_LENGTH_ZERO
)
#define IDT_HARC_KEY_QUEUE_OVERFLOW     (HLLAPI_ERROR_BASE + HARC_KEY_QUEUE_OVERFLOW
)

#define HLLAPI_ERROR_LIMIT      IDT_HARC_KEY_QUEUE_OVERFLOW
#define IDT_UNKNOWN_ERR         HLLAPI_ERROR_LIMIT + 1

BOOL InitApplication(HANDLE);
BOOL InitInstance(HANDLE, int);
void InvokeFunction(WORD iFuncNum, LPSTR szDataString, WORD iDataStrLen, WORD
iPSPosition);
int  ParseString(LPBYTE dst, LPBYTE src, int  count);
void OutputTextString(LPBYTE lpOutString);
void ScrollScreenRows(int nVscrollInc);
int  GetNameFromFuncNum(WORD FuncNumber, LPSTR FuncName, WORD *OutLogAvail);
void GetErrorFromErnum(int ErrorNumber, LPSTR ErrorName);
void OutputFormatData(LPCSTR lpAPIDataString, WORD DataStringLen);

int  WINAPI WinMain(HANDLE, HANDLE, LPSTR, int);
long WINAPI MainWndProc(HWND, unsigned, WPARAM, LPARAM);
BOOL WINAPI FuncDlgProc(HWND, UINT, WPARAM, LPARAM);
BOOL WINAPI AboutDlgProc(HWND, UINT, WPARAM, LPARAM);
BOOL WINAPI ShutDownDlgProc(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam);


/*******************************\
*                               *
* Windows 16 -> 32 bit          *
*                               *
* Portability issues            *
*                               *
\*******************************/
#if defined WIN32   // Following section for Win32

#ifndef MEMCPY
#define MEMCPY      memcpy
#endif
#ifndef MEMSET
#define MEMSET      memset
#endif


#else               // Following section for Windows 16 bit versions
#ifndef LPBYTE
typedef BYTE FAR *     LPBYTE;
#endif

#ifndef MEMCPY
#define MEMCPY      _fmemcpy
```

```
#endif
#ifndef MEMSET
#define MEMSET      _fmemset
#endif
                    // WM_COMMAND support macros (in windowsx.h for win32)
                    // Although documented as being in Windowsx.h for 16-bit
                    // they do not seem to exist.
#define GET_WM_COMMAND_ID(wp, lp)               wp
#define GET_WM_COMMAND_HWND(wp, lp)             (HWND)(LOWORD(lp))
#define GET_WM_COMMAND_CMD(wp, lp)              HIWORD(lp)
#define GET_WM_COMMAND_MPS(id, hwnd, cmd)    \
        (WPARAM)(UINT)(id), MAKELPARAM((UINT)hwnd, cmd)


#endif

#if !defined WIN32
#define IDS_VERS_APPNAME    "HAPITEST\0"        /* name of the application */
#else
#define IDS_VERS_APPNAME    "APITST32\0"        /* name of the application */
#endif


#define IDS_VERSNUM_VALUE   3,03,00
#define IDS_VERSNUM_STR     "3.3.0\0"
```

MAKEFILE (Windows)

```
all: hapitest.exe

!if "$(DEBUG)" == "yes"
COMPIL=-c -Gsw -FPa -W2 -Od -Zip -AM
LINK=/M/CO/NOP/NOD/NOE
!else
COMPIL=-c -Gsw -FPa -W2 -Os -Zp -AM
LINK=/M/NOP/NOD/NOE
!endif

# Update the resource if necessary

hapitest.res: hapitest.rc hapitest.dlg hapitest.h
    rc -r hapitest.rc

# Update the object file if necessary

hapitest.obj: hapitest.c hapitest.h
    set CL=$(COMPIL)
    cl hapitest.c

# Update the executable file if necessary, and if so, add the resource back in.

hapitest.exe: hapitest.obj hapitest.def hapitest.res
    del *.bak
    link $(LINK) hapitest,hapitest,, libw mlibcew, hapitest.def
    rc hapitest.res
```

MAKEFILE (Windows NT)

```
!include <ntwin32.mak>

all: apitst32.exe

srcpth  =   \win32app\ttlkw32\apitst32

!if "$(DEBUG)" == "yes"
COMPIL=$(cc) $(cflags) $(cvarsdll) $(cdebug) /DBUSE3D /DDOS
LINKOPT=$(link) $(linkdebug) $(guiflags) $(linkdebug) -out:apitst32.exe /
                                                    MAP:apitst32.map $(guilibsdll)
!else
COMPIL=$(cc) $(cflags) $(cvarsdll) /DBUSE3D /DDOS
LINKOPT=$(link) $(guiflags) -out:apitst32.exe /MAP:apitst32.map $(guilibsdll)
!endif

INCLU=/I $(srcpth)
RCINCLU=/I $(srcpth)
# Update the resource if necessary

apitst32.res: $(srcpth)\hapitest.rc  \
        $(srcpth)\hapitest.dlg \
        $(srcpth)\hapitest.h
    rc -r $(RCINCLU) -foapitst32.res $(srcpth)\hapitest.rc

apitst32.rbj: tt4w32hapi.res
      cvtres -$(CPU) apitst32.res -o apitst32.rbj

apitst32.obj: $(srcpth)\hapitest.c
     $(COMPIL) $(INCLU) /Foapitst32.obj $(srcpth)\hapitest.c > hapi4w32.err

apitst32.exe: apitst32.obj apitst32.rbj
    $(LINKOPT) apitst32.obj  apitst32.rbj \
    wsock32.lib netapi32.lib $(crtdll) ctl3d32.lib \
       -out:apitst32.exe -map:apitst32.map \
    $(srcpth)\pcshll32.lib >> hapi4w32.err
```

HAPITEST.DEF

```
; module-definition file for generic – used by LINK.EXE

NAME         HAPITEST      ; application's module name

DESCRIPTION  'HAPITEST from Pericom Software'

EXETYPE      WINDOWS    ; required for all Windows applications

STUB      'WINSTUB.EXE' ; Generates error message if application
               ; is run without Windows

;CODE can be moved in memory and discarded/reloaded
CODE  PRELOAD MOVEABLE DISCARDABLE

;DATA must be MULTIPLE if program can be invoked more than once
DATA  PRELOAD MOVEABLE MULTIPLE


HEAPSIZE     1024
STACKSIZE    5120          ; recommended minimum for Windows applications


; All functions that will be called by any Windows routine
; MUST be exported.

EXPORTS
        MainWndProc         @1   ; name of window processing function
        FuncDlgProc         @2
        AboutDlgProc        @3
        ShutDownDlgProc     @4

IMPORTS
        PCSHLL.hllapi
        PCSHLL.open_workstation
        PCSHLL.close_workstation
```

HAPITEST.RC

```
#include <windows.h>
#include <ver.h>
#include “hapitest.h”
#include “hapifunc.h”

HAPIICON  ICON    hapitest.ico

HapiTestMenu MENU
BEGIN
    POPUP        “&File”
    BEGIN
        MENUITEM “Exit...”,                    IDM_TEST_EXIT
    END

    POPUP        “&Options”
    BEGIN
        MENUITEM “&Invoke HLLAPI Function...”, IDM_FUNCTION
        MENUITEM “&Clear Buffer”,              IDM_CLEAR
    END

    POPUP        “&Help”
    BEGIN
        MENUITEM “&About Tester...”,           IDM_ABOUT
    END
END


STRINGTABLE
BEGIN

/* Function Names 3000-> */

IDT_CONNECT_PS              “Connect Presentation Space”
IDT_DISCONNECT_PS           “Disconnect Presentation Space”
IDT_SEND_KEYSTRING          “Send Keystring”
IDT_WAIT                    “Wait”
IDT_COPY_PS                 “Copy Presentation Space”
IDT_SEARCH_PS               “Search Presentation Space”
IDT_QUERY_CURSOR_LOC        “Query Cursor Location”
IDT_COPY_PS_TO_STRING       “Copy Presentation Space to String”
IDT_SET_SESSION_PARAMS      “Set Session Parameters”
IDT_QUERY_SESSIONS          “Query Sessions”
IDT_RESERVE                 “Reserve”
IDT_RELEASE                 “Release”
IDT_COPY_OIA                “Copy OIA”
IDT_QUERY_FIELD_ATTR        “Query Field Attribute”
IDT_COPY_STRING_TO_PS       “Copy String To Presentation Space”
IDT_FUNC_16                 “”
IDT_FUNC_17                 “”
IDT_PAUSE                   “Pause”
IDT_FUNC_19                 “”
IDT_QUERY_SYSTEM            “Query System”
```

```
IDT_RESET_SYSTEM              "Reset System"
IDT_QUERY_SESSION_STATUS      "Query Session Status"
IDT_START_HOST_NOTIF          "Start Host Notification"
IDT_QUERY_HOST_UPDATE         "Query Host Update"
IDT_STOP_HOST_NOTIF           "Stop Host Notification"
IDT_FUNC_26                   ""
IDT_FUNC_27                   ""
IDT_FUNC_28                   ""
IDT_FUNC_29                   ""
IDT_SEARCH_FIELD              "Search Field"
IDT_FIND_FIELD_POSITION       "Find Field Position"
IDT_FIND_FIELD_LENGTH         "Find Field Length"
IDT_COPY_STRING_TO_FIELD      "Copy String To Field"
IDT_COPY_FIELD_TO_STRING      "Copy Field To String"
IDT_FUNC_35                   ""
IDT_FUNC_36                   ""
IDT_FUNC_37                   ""
IDT_FUNC_38                   ""
IDT_FUNC_39                   ""
IDT_SET_CURSOR                "Set Cursor"
IDT_FUNC_41                   ""
IDT_FUNC_42                   ""
IDT_FUNC_43                   ""
IDT_FUNC_44                   ""
IDT_FUNC_45                   ""
IDT_FUNC_46                   ""
IDT_FUNC_47                   ""
IDT_FUNC_48                   ""
IDT_FUNC_49                   ""
IDT_START_KEY_INTERCEPT       "Start Key Intercept"
IDT_GETKEY                    "Get Key"
IDT_POST_INTERCEPT_STATUS     "Post Intercept Status"
IDT_STOP_KEY_INTERCEPT        "Stop Key Intercept"
IDT_FUNC_54                   ""
IDT_FUNC_55                   ""
IDT_FUNC_56                   ""
IDT_FUNC_57                   ""
IDT_FUNC_58                   ""
IDT_FUNC_59                   ""
IDT_FUNC_60                   ""
IDT_FUNC_61                   ""
IDT_FUNC_62                   ""
IDT_FUNC_63                   ""
IDT_FUNC_64                   ""
IDT_FUNC_65                   ""
IDT_FUNC_66                   ""
IDT_FUNC_67                   ""
IDT_FUNC_68                   ""
IDT_FUNC_69                   ""
IDT_FUNC_70                   ""
IDT_FUNC_71                   ""
IDT_FUNC_72                   ""
IDT_FUNC_73                   ""
IDT_FUNC_74                   ""
IDT_FUNC_75                   ""
```

```
IDT_FUNC_76                 ""
IDT_FUNC_77                 ""
IDT_FUNC_78                 ""
IDT_FUNC_79                 ""
IDT_FUNC_80                 ""
IDT_FUNC_81                 ""
IDT_FUNC_82                 ""
IDT_FUNC_83                 ""
IDT_FUNC_84                 ""
IDT_FUNC_85                 ""
IDT_FUNC_86                 ""
IDT_FUNC_87                 ""
IDT_FUNC_88                 ""
IDT_FUNC_89                 ""
IDT_FUNC_90                 ""
IDT_FUNC_91                 ""
IDT_FUNC_92                 ""
IDT_FUNC_93                 ""
IDT_FUNC_94                 ""
IDT_FUNC_95                 ""
IDT_FUNC_96                 ""
IDT_FUNC_97                 ""
IDT_FUNC_98                 ""
IDT_CONVERT_POSITION        "Convert Position / RowCol"
IDT_FUNC_100                ""
IDT_FUNC_101                ""
IDT_FUNC_102                ""
IDT_SEND_KEYPRESS           "Send KeyPress"
IDT_FUNC_104                ""
IDT_COPY_FORM_PS            "Copy Formatted PS"
IDT_FUNC_106                ""
IDT_FUNC_107                ""
IDT_COPY_FORM_PS_TO_STRING  "Copy Formatted PS To String"
IDT_SET_TEEMTALK_PARAM      "Set Teemtalk Parameter"
IDT_GET_TEEMTALK_PARAM      "Get Teemtalk Parameter"
IDT_FUNC_111                ""
IDT_FUNC_112                ""
IDT_FUNC_113                ""
IDT_QUERY_FORM_FIELD_ATTR   "Query Formatted Field Attribute"
IDT_FUNC_115                ""
IDT_FUNC_116                ""
IDT_FUNC_117                ""
IDT_FUNC_118                ""
IDT_FUNC_119                ""
IDT_FUNC_120                ""
IDT_FUNC_121                ""
IDT_FUNC_122                ""
IDT_FUNC_123                ""
IDT_FUNC_124                ""
IDT_FUNC_125                ""
IDT_FUNC_126                ""
IDT_FUNC_127                ""
IDT_PERICOM_FUNCTIONS       "Pericom Functions"
IDT_HOST_INPUT              "Host Input"
IDT_WINDOW_HANDLE           "Window Handle"
```

```
IDT_WS_OPEN                  “Open Workstation”
IDT_WS_CLOSE                 “Close Workstation”

#if !defined WIN32
IDT_TEST_PROGNAME           “Api Tester Program (16 Bit)”
#else
IDT_TEST_PROGNAME           “Api Tester Program (32 Bit)”
#endif

/* SYSTEM ERRORS 4000-> */

IDT_HARC_SUCCESS                “Command Successful”
IDT_HARC_INVALID_PS             “Not Currently Connected”
IDT_HARC_PARAM_ERROR            “Parameter Error”
IDT_HARC_BUSY_PS                “Presentation Space Busy”
IDT_HARC_FUNCTION_INHIBITED     “Presentation Space Locked”
IDT_HARC_DATA_ERROR             “Data Size Error”
IDT_HARC_INVALID_PS_POSITION    “PS Position Invalid”
IDT_HARC_NO_PREREQ_ISSUED       “Prerequired Function”
IDT_HARC_SYSTEM_ERROR           “System Error”
IDT_HARC_FUNC_NOT_SUPPORTED     “Function Not Supported”
IDT_HARC_RESOURCE_UNAVAIL       “Resource Unavailable”
IDT_HARC_INVALID_KEYSTROKE      “Invalid Keystroke”
IDT_HARC_OIA_UPDATED            “OIA Updated”
IDT_HARC_PS_UPDATED             “PS Updated”
IDT_HARC_OIA_AND_PS_UPDATED     “OIA & PS Updated”
IDT_HARC_NOT_FOUND              “Not Found / PS Unformatted”
IDT_HARC_KEYS_NOT_AVAIL         “Keystrokes Not Available”
IDT_HARC_HOST_EVENT_OCCURED     “PS or OIA Updated”
IDT_HARC_FIXED_LENGTH_ZERO      “Field Length was Zero”
IDT_HARC_KEY_QUEUE_OVERFLOW     “Keystroke Queue Overflow”

IDT_UNKNOWN_ERR                 “Invalid Return”

END

#include “hapitest.dlg”


/////////////////////////////////////////////////////////////////////////////
//
// Version
//


VS_VERSION_INFO VERSIONINFO
 FILEVERSION    IDS_VERSNUM_VALUE
 PRODUCTVERSION IDS_VERSNUM_VALUE
 FILEFLAGSMASK  VS_FFI_FILEFLAGSMASK
#ifdef _DEBUG
 FILEFLAGS      VS_FF_DEBUG
#else
 FILEFLAGS      0x0L
#endif
#if !defined WIN32
```

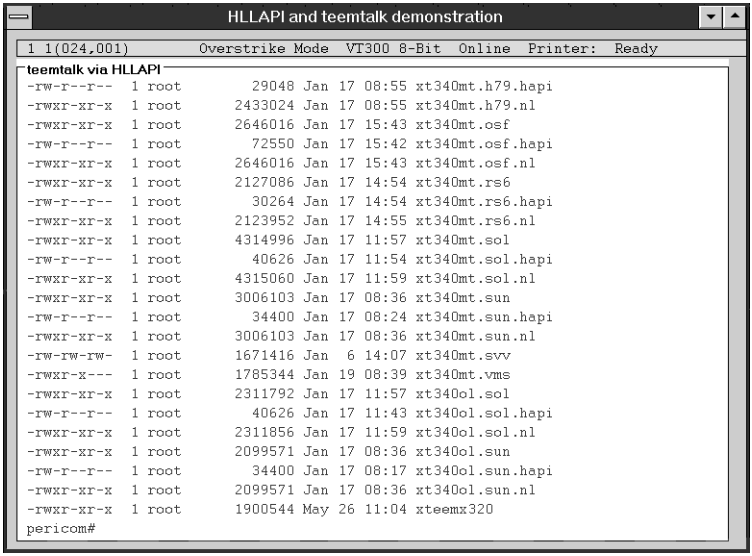
```
 FILEOS          VOS_DOS_WINDOWS16
#else
 FILEOS          VOS__WINDOWS32
#endif
 FILETYPE        VFT_APP
 FILESUBTYPE     VFT2_UNKNOWN
BEGIN
    BLOCK “StringFileInfo”
    BEGIN
        BLOCK “080904e4”
        BEGIN
            VALUE “CompanyName”,      “Pericom Software Plc.\0”
#if !defined WIN32
            VALUE “Comments”,         “teemtalk for Windows\0”
            VALUE “FileDescription”,  “teemtalk for Windows HLLAPI test\0”
            VALUE “ProductName”,      “teemtalk for Windows\0”
#else
            VALUE “Comments”,         “teemtalk for Windows 32\0”
            VALUE “FileDescription”,  “teemtalk for Windows 32 HLLAPI test\0”
            VALUE “ProductName”,      “teemtalk for Windows 32\0”
#endif
            VALUE “FileVersion”,      IDS_VERSNUM_STR
            VALUE “InternalName”,     IDS_VERS_APPNAME
            VALUE “LegalCopyright”,   “Copyright \251 Pericom Software Plc, 1995-
1997\0”
            VALUE “LegalTrademarks”,  “Microsoft\256 is a registered trademark of
Microsoft Corporation. teemtalk\256 is a registered trademark of Pericom Software
Plc.\0”
            VALUE “OriginalFilename”, “\0”
            VALUE “ProductVersion”,   IDS_VERSNUM_STR
        END
    END
    BLOCK “VarFileInfo”
    BEGIN
        VALUE “Translation”, 0x809, 1252
    END
END
```

HAPITEST.DLG

```
DLGINCLUDE RCDATA DISCARDABLE
BEGIN
    “hapitest.H\0”
END

ABOUTBOX DIALOG 22, 17, 144, 75
STYLE DS_MODALFRAME | WS_CAPTION | WS_SYSMENU
CAPTION “About Tester”
FONT 8, “Helv”
BEGIN
    CTEXT           “”, IDD_TEST_PROGNAME, 0, 4, 144, 8
    CTEXT           “ from Pericom Software Plc.”, -1, 0, 16, 144, 8
    CTEXT           “”, IDD_TEST_VERSNUM, 0, 47, 144, 8
    ICON            “HAPIICON”, IDD_TEST_ICON, 62, 26, 16, 16, WS_GROUP
    DEFPUSHBUTTON   “OK”, IDOK, 53, 59, 32, 14, WS_GROUP
END

SHUTDOWN DIALOG 22, 17, 177, 47
STYLE WS_POPUP | WS_CAPTION
FONT 8, “Helv”
BEGIN
    CTEXT           “”, IDD_TEST_SHUTDOWN, 23, 10, 154, 9, NOT WS_GROUP
    ICON            “HAPIICON”, IDD_TEST_ICON, 4, 4, 21, 20
    PUSHBUTTON      “”, IDOK, 33, 26, 39, 14, WS_GROUP
    PUSHBUTTON      “”, IDCANCEL, 111, 26, 40, 14, NOT WS_TABSTOP
END

FUNCTIONBOX DIALOG 146, 61, 160, 147
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION “Function Call”
FONT 8, “Helv”
BEGIN
    LTEXT           “Function”, -1, 2, 9, 33, 12
    COMBOBOX        IDD_TEST_FUNCNUMBER, 36, 9, 120, 67, CBS_DROPDOWNLIST |
                    WS_VSCROLL | WS_TABSTOP
    GROUPBOX        “Data Packet”, -1, 0, 22, 159, 45, WS_GROUP
    LTEXT           “Session Identifier”, -1, 2, 34, 66, 15, NOT WS_GROUP
    LTEXT           “Data “, -1, 3, 53, 33, 10, NOT WS_GROUP
    COMBOBOX        IDD_TEST_SESSID, 69, 32, 87, 35, CBS_DROPDOWNLIST |
                    WS_VSCROLL | WS_TABSTOP
    EDITTEXT        IDD_TEST_DATASTR, 37, 51, 119, 12, ES_AUTOHSCROLL
    GROUPBOX        “Data Packet Length”, -1, 0, 71, 159, 33, WS_GROUP
    LTEXT           “String Length”, -1, 2, 82, 31, 18, NOT WS_GROUP
    EDITTEXT        IDD_TEST_STRLEN, 39, 85, 30, 12, ES_AUTOHSCROLL
    PUSHBUTTON      “Ca&lculate”, IDD_TEST_STRCALC, 99, 84, 40, 14
    LTEXT           “PS. Position”, -1, 2, 108, 31, 18
    EDITTEXT        IDD_TEST_PSPOSN, 39, 111, 30, 12, ES_AUTOHSCROLL
    DEFPUSHBUTTON   “O&K”, IDOK, 18, 129, 39, 14, WS_GROUP
    PUSHBUTTON      “Ca&ncel”, IDCANCEL, 99, 128, 39, 14, NOT WS_TABSTOP
END
```

Visual Basic API Example

TTLINK is a Visual Basic application. It creates a HLLAPI link to *teemtalk* and shows the user the *teemtalk* screen. When the application is run you should see a window similar to the following:



The user cannot input data into this window. There are two files that make up this example: **APILINK.BAS** and **TTLINK.FRM**.

> *Note: To make an executable file from Visual Basic, select **Make EXE File...** in the **File** menu and enter a suitable file name.*

APILINK.BAS

The file **APILINK.BAS** contains the main code. The first section here sets up the application to use **PCSHLL.DLL**.

```
Declare Function hllapi Lib “pcshll” (iFuncNum%, ByVal LpDataString$, iStringLen%,
iRetCode%) As Integer

Global iFinished%

Sub Main ()

Dim Updated%
Dim ApiFunc%, ApiStrLen%, ApiRetCode%
Dim ApiString As String * 2000
Dim ApiOIA As String * 300
Dim RetCode%
Dim i%

    iFinished = 0


    ApiFunc = 1
    ApiString = “A”
    RetCode = hllapi(ApiFunc, ApiString, ApiStrLen, ApiRetCode)

    ApiFunc = 9
    ApiString = “IPAUSE”
    ApiStrLen = Len(ApiString)
    RetCode = hllapi(ApiFunc, ApiString, ApiStrLen, ApiRetCode)

    ApiFunc = 23
    ApiString = “AB”
    RetCode = hllapi(ApiFunc, ApiString, ApiStrLen, ApiRetCode)

    ttlink.Show

    Do
        Updated = 0

        ApiFunc = 5
        RetCode = hllapi(ApiFunc, ApiString, ApiStrLen, ApiRetCode)

        ApiFunc = 13
        ApiStrLen = 103
        RetCode = hllapi(ApiFunc, ApiOIA, ApiStrLen, ApiRetCode)

        For i = 0 To 23
            ttlink.scrline(i) = Mid$(ApiString, ((i * 80) + 1), 80)
        Next i
        ttlink.status1 = Mid$(ApiOIA, 3, 80)

        Do
```

```
            DoEvents

            ApiFunc = 18
            ApiStrLen = 2
            RetCode = hllapi(ApiFunc, ApiString, ApiStrLen, ApiRetCode)

            If (Not iFinished) Then
                ApiFunc = 24
                ApiString = "A"
                RetCode = hllapi(ApiFunc, ApiString, ApiStrLen, ApiRetCode)
                If (ApiRetCode > 9) Then
                    Updated = 1
                End If
            End If
        Loop Until (Updated + iFinished)

    Loop Until iFinished

    ApiFunc = 25
    ApiString = "A"
    RetCode = hllapi(ApiFunc, ApiString, ApiStrLen, ApiRetCode)

    ApiFunc = 2
    ApiString = "A"
    RetCode = hllapi(ApiFunc, ApiString, ApiStrLen, ApiRetCode)

End Sub
```

TTLINK.FRM

The **TTLINK.FRM** file sets up the form and creates an EXIT button which closes the application.

```
VERSION 2.00
Begin Form ttlink
   BackColor       =   &H00C0C0C0&
   Caption         =   "HLLAPI and teemtalk demonstration"
   ClientHeight    =   6792
   ClientLeft      =   264
   ClientTop       =   276
   ClientWidth     =   9228
   FontBold        =   -1  'True
   FontItalic      =   0   'False
   FontName        =   "Courier"
   FontSize        =   9.6
   FontStrikethru  =   0   'False
   FontUnderline   =   0   'False
   Height          =   7212
   Left            =   216
   LinkTopic       =   "Form1"
```

```
   ScaleHeight     =   6792
   ScaleWidth      =   9228
   Top             =   -96
   Width           =   9324
   Begin CommandButton Command1
      Cancel          =   -1  'True
      Caption         =   "Exit"
      Height          =   372
      Left            =   3360
      TabIndex        =   0
      Top             =   6360
      Width           =   2172
   End
   Begin Frame Frame1
      BackColor       =   &H0000FFFF&
      Caption         =   "teemtalk via HLLAPI"
      Height          =   6012
      Left            =   120
      TabIndex        =   1
      Top             =   360
      Width           =   8892
      Begin Label scrline
         AutoSize        =   -1  'True
         BackColor       =   &H0000FFFF&
         FontBold        =   0   'False
         FontItalic      =   0   'False
         FontName        =   "Courier"
         FontSize        =   9.6
         FontStrikethru  =   0   'False
         FontUnderline   =   0   'False
         ForeColor       =   &H00C00000&
         Height          =   192
         Index           =   23
         Left            =   120
         TabIndex        =   3
         Top             =   5760
         Width           =   8700
         WordWrap        =   -1  'True
      End
```

The highlighted section above is repeated with the following changed values:

```
Index    22  21  20  19  18  17  16  15  14  13  12  11  10  9   8   7   6   5   4   3
         2   1   0
TabIndex 26  25  24  23  22  21  20  19  18  17  16  15  14  13  12  11  10  9   8
             7   6   5   4
Top   5520  5280  5040  4800  4560  4320  4080  3840  3600  3360  3120  2880  2640  2400
      2160  1920  1680  1440  1200  960   720   480   240
```

```
   Begin Label status1
      AutoSize        =   -1  'True
      BackColor       =   &H0000FF00&
      BorderStyle     =   1  'Fixed Single
      FontBold        =   0   'False
```

```
      FontItalic      =   0   'False
      FontName        =   "Courier"
      FontSize        =   9.6
      FontStrikethru  =   0   'False
      FontUnderline   =   0   'False
      ForeColor       =   &H000000FF&
      Height          =   216
      Left            =   120
      TabIndex        =   2
      Top             =   120
      Width           =   8916
      WordWrap        =   -1  'True
   End
End

Sub Command1_Click ()
    'Unload form
    Unload ttlink
    iFinished = 1
End Sub
```

Notes

C

*teem*X *API Program Examples*

This appendix shows example API programs for use with *teem*X.

Introduction

The Programmer's API Development Kit includes the following example API program files, the contents of which are shown on the following pages.

HAPIFUNC.H	Defines the functions for *teem*X (and *teemtalk*) API programs.
APITEST.C	Main routines.

HAPIFUNC.H

```
/*****************************************************************\
*                                                                 *
*                                                                 *
*            Copyright (C) 1989-1996 Pericom Software PLC         *
*                        All Rights Reserved                      *
*                                                                 *
*                       Pericom Software PLC.                     *
*                            The Priory                           *
*                             Cosgrove                            *
*                           Milton Keynes                         *
*                          Bucks, MK19 7JJ                        *
*                              England                            *
*                                                                 *
\*****************************************************************/


/***************************/
/* HLLAPI Function Numbers */
/***************************/

#define HA_CONNECT_PS                1
#define HA_DISCONNECT_PS             2
#define HA_SEND_KEYSTRING            3
#define HA_WAIT                      4
#define HA_COPY_PS                   5
#define HA_SEARCH_PS                 6
#define HA_QUERY_CURSOR_LOC          7
#define HA_COPY_PS_TO_STRING         8
#define HA_SET_SESSION_PARAMS        9
#define HA_QUERY_SESSIONS           10
#define HA_RESERVE                  11
#define HA_RELEASE                  12
#define HA_COPY_OIA                 13
#define HA_QUERY_FIELD_ATTR         14
#define HA_COPY_STRING_TO_PS        15
/* not used 16 */
#define HA_STORAGE_MANAGER          17
#define HA_PAUSE                    18
/* not used 19 */
#define HA_QUERY_SYSTEM             20
#define HA_RESET_SYSTEM             21
#define HA_QUERY_SESSION_STATUS     22
#define HA_START_HOST_NOTIF         23
#define HA_QUERY_HOST_UPDATE        24
#define HA_STOP_HOST_NOTIF          25
/* not used 26 */
/* not used 27 */
/* not used 28 */
/* not used 29 */
#define HA_SEARCH_FIELD             30
#define HA_FIND_FIELD_POSITION      31
```

```
#define HA_FIND_FIELD_LENGTH         32
#define HA_COPY_STRING_TO_FIELD      33
#define HA_COPY_FIELD_TO_STRING      34
/* not used 35 */
/* not used 36 */
/* not used 37 */
/* not used 38 */
/* not used 39 */
#define HA_SET_CURSOR                40
/* not used 41 */
/* not used 42 */
/* not used 43 */
/* not used 44 */
/* not used 45 */
/* not used 46 */
/* not used 47 */
/* not used 48 */
/* not used 49 */
#define HA_START_KEY_INTERCEPT       50
#define HA_GETKEY                    51
#define HA_POST_INTERCEPT_STAT       52
#define HA_STOP_KEY_INTERCEPT        53
/* not used 54 to 89 inclusive */
#define HA_SEND_FILE                 90
#define HA_RECEIVE_FILE              91
/* not used 92 */
/* not used 93 */
/* not used 94 */
/* not used 95 */
/* not used 96 */
/* not used 97 */
/* not used 98 */
#define HA_CONVERT_POSITION          99
#define HA_CONVERT_ROWCOL            99
/* not used 100 */
/* not used 101 */
/* not used 102 */
#define HA_SEND_KEYPRESS            103
/* not used 104 */
#define HA_COPY_FORM_PS             105
/* not used 106 */
/* not usd 107 */
#define HA_COPY_FORM_PS_TO_STRING   108
#define HA_SET_TEEMTALK_PARAM       109
#define HA_GET_TEEMTALK_PARAM       110
/* not used 111 */
/* not used 112 */
/* not used 113 */
#define HA_QUERY_FORM_FIELD_ATTR    114
/* not used 115 */
/* not used 116 */
/* not used 117 */
/* not used 118 */
/* not used 119 */
/* not used 120 */
/* not used 121 */
```

```
/* not used 122 */
/* not used 123 */
/* not used 124 */
/* not used 125 */
/* not used 126 */
/* not used 127 */
#define HA_PERICOM_FUNCTIONS        128
#define HA_HOST_INPUT               129
#define HA_WINDOW_HANDLE            130


/************************************************/
/* HLLAPI Pericom Functions (128) Function Numbers */
/************************************************/
#define HA_PERIFUNC_SAVESET     1


/***********************/
/* HLLAPI Return Values */
/***********************/

#define HA_STATUS_CODE_INVALID_PS   9998
#define HA_STATUS_CODE_PARAM_ERROR  9999

#define HARC_SUCCESS                 0
#define HARC_NO_HOST_UPDATES         0
#define HARC_INVALID_PS              1
#define HARC_PARAM_ERROR             2
/* not used 3 */
#define HARC_BUSY_PS                 4
#define HARC_FUNCTION_INHIBITED      5
#define HARC_ILLEGAL_DATA            5
#define HARC_DATA_ERROR              6
#define HARC_INVALID_PS_POSITION     7
#define HARC_NO_PREREQ_ISSUED        8
#define HARC_SYSTEM_ERROR            9
#define HARC_FUNC_NOT_SUPPORTED     10
#define HARC_RESOURCE_UNAVAIL       11
/* not used 12 */
/* not used 13 */
/* not used 14 */
/* not used 15 */
/* not used 16 */
/* not used 17 */
/* not used 18 */
/* not used 19 */
#define HARC_INVALID_KEYSTROKE      20
#define HARC_OIA_UPDATED            21
#define HARC_PS_UPDATED             22
#define HARC_OIA_AND_PS_UPDATED     23
#define HARC_NOT_FOUND              24
#define HARC_PS_UNFORMATTED         24
#define HARC_KEYS_NOT_AVAIL         25
#define HARC_HOST_EVENT_OCCURED     26
/* not used 27 */
#define HARC_FIXED_LENGTH_ZERO      28
```

```
/* not used 29 */
/* not used 30 */
#define HARC_KEY_QUEUE_OVERFLOW     31


/*****************************/
/* HLLAPI Extern declarations */
/*****************************/
#ifdef __cplusplus
extern "C" {            /* Assume C declarations for C++ */
#endif   /* __cplusplus */
#ifdef WINVER
#ifdef WIN32
__declspec(dllexport)
WORD    WINAPI  hllapi(
                    LPWORD  lpwFuncNum,
                    LPBYTE  lpDataString,
                    LPWORD  lpwDataStringLength,
                    LPWORD  lpwRetCode
                    );
__declspec(dllexport)
int     WINAPI  open_workstation(
                    int     iShortname,
                    LPSTR   lpArgs
                    );
__declspec(dllexport)
int     WINAPI  close_workstation(
                    int     iShortname
                    );
#else
int FAR PASCAL  hllapi(
                    LPINT   lpiFuncNum,
                    LPSTR   lpDataString,
                    LPINT   lpiDataStringLength,
                    LPINT   lpiRetCode
                    );
int     WINAPI  open_workstation(
                    int     iShortname,
                    LPSTR   lpArgs
                    );
int     WINAPI  close_workstation(
                    int     iShortname
                    );
#endif
#else
int             hllc(
                 /* int     *function_number,       */
                 /* char    *data_string,           */
                 /* int     *data_string_length,    */
                 /* int     *return_code            */
                    );
int             open_workstation(
                 /* int     iShortname,             */
                 /* char    *lpArgs                 */
                    );
```

```
int             close_workstation(
                 /* int     iShortname              */
                    );
#endif
#ifdef __cplusplus
}                       /* End of extern “C” { */
#endif   /* __cplusplus */
```

APITEST.C

```
/*************************************************************
 *
 *  NAME
 *    apitest.c
 *
 *  DESCRIPTION
 *    This function provides 'APItest' terminal emulation
 *
 *************************************************************/
#include "hapifunc.h"
#include <stdio.h>


main(argc,argv)
int argc;
char *argv[];
{

int api_func;
char api_str[40000];
int api_len;
int api_ps;
int rc;
int cr_req;
char *cr;


cr = "\r";

for ( ; ; )
  {
  printf("Enter fc len ps cr? and then return to enter string:\n");
  scanf("%d %d %d %d",&api_func,&api_len,&api_ps,&cr_req);
  printf("string:\n");

if (cr_reg == 2)
     break;

  rc = read(0,api_str,2048);

  if(rc == 0)
      api_str[0] = '\0';
  else
      api_str[rc-1]= '\0';

  if(cr_req == 1)
         strcat(api_str,cr,2);

  if(api_len == 0)
      api_len = strlen(api_str);
```

```
  rc = hllc(&api_func,api_str,&api_len,&api_ps);

  if(rc != 0)
      printf("Bad return code = %d \n",rc);
      else
      printf("fc = %d, len = %d, ps = %d data next
line:\n%s\n",api_func,api_len,api_ps,api_str);

}
```

Index

A

C

D

E

F

G

H

I

K

L

M

O

P

Q

R

S

T

U

V

W