

Emulator Programming



About This Book

This book provides necessary programming information for you to use the HCL Z and I Emulator for Windows Emulator High-Level Language Application Program Interface (EHLLAPI), and Z and I Emulator for Windows Session API (PCSAPI), and. The Host Access Class Library is described in *Host Access Class Library*.

EHLLAPI/PCSAPI is used with Z and I Emulator for Windows to provide a way for users and programmers to access the host presentation space with a set of functions that can be called from an application program running in a workstation session.

In this book, *Windows* refers to Windows® 7, Windows® 8/8.1, Windows® 10, Windows® Server 2008, and Windows® Server 2012. When information is relevant only to a specific operating system, this will be indicated in the text.

Who Should Read This Book

This book is intended for programmers who write application programs that use the APIs documented in this book.

A working knowledge of Windows® is assumed. For information about Windows®, refer to the list of publications under [Where To Find More Information on page ii](#).

The programmer must also be familiar with connecting to a host system from a terminal or from a workstation with terminal emulation software.

This book assumes you are familiar with the language and the compiler that you are using. For information on how to write, compile, or link-edit programs, refer to [Where To Find More Information on page ii](#) for the appropriate references for the specific language you are using.

Where To Find More Information

The Z and I Emulator for Windows library includes the following publications:

- *Installation Guide*
- *Quick Beginnings*
- *Emulator User's Reference*
- *Administrator's Guide and Reference*
- *Emulator Programming*
- *Client/Server Communications Programming*
- *System Management Programming*
- *Host Access Class Library*
- *Configuration File Reference*

In addition to the printed books, there are Hypertext Markup Language (HTML) documents provided with Z and I Emulator for Windows:

Host Access Class Library

The HACL Java HTML files describe how to write an ActiveX/OLE 2.0-compliant application to use Z and I Emulator for Windows as an embedded object. These files can be accessed from the Docs_Admin_Aids zipped folder delivered along with Z and I Emulator for Windows product documentation in the following path : *ZIEWin_3.0_Docs_Admin_Aids.zip\publications\en_US\doc\hac*

Following is a list of related publications:

- *IBM 3270 Information Display System Data Stream Programmer's Reference*, GA23-0059
- *IBM 5250 Information Display System Functions Reference Manual*, SA21-9247

Notation

A table at the beginning of each section explains API functions in [EHLLAPI Functions on page 31](#), [PCSAPI Functions on page 188](#), and [WinHLLAPI Extension Functions on page 175](#). It shows whether a function is supported for the products that provide the function described in the section. Yes means it is supported for a host type, and No means not supported. For example, the following table indicates that a function is available for 3270 and VT sessions but not for 5250 sessions.

<i>3270</i>	<i>5250</i>	<i>VT</i>
Yes	No	Yes

Chapter 1. Introduction to Emulator APIs

The IBM® Z and I Emulator for Windows product supplies several application programming interfaces (APIs). Each interface has a specific set of functions and may be used for different purposes. Choose the programming interface that best matches the functional requirements of your application. Some applications may use more than one interface to achieve the desired results. The programming interfaces are:

- **Emulator High Level Language API (EHLLAPI):** This interface provides functions to access emulator "presentation space" data such as characters on the host screen. It also provides functions for sending keystrokes to the host, intercepting user-entered keystrokes, querying the status of the host session, uploading and downloading files, and other functions. This interface is often used for *automated operator* applications which read host screens and enter keystrokes without direct user intervention. See [EHLLAPI Functions on page 31](#).
 - **IBM® Standard HLLAPI Support:** This is a standard programming interface which allows programmatic access to a host emulator session. See [Introduction to IBM Standard EHLLAPI, IBM Enhanced EHLLAPI and WinHLLAPI Programming on page 7](#).
 - **IBM® Enhanced HLLAPI Support:** This interface is based on the IBM® Standard HLLAPI interface. It provides all of the existing functionality but uses modified data structures. See [Introduction to IBM Standard EHLLAPI, IBM Enhanced EHLLAPI and WinHLLAPI Programming on page 7](#).
 - **Windows® High Level Language API (WinHLLAPI):** This interface provides much of the same functionality of IBM® Standard EHLLAPI and adds some extensions that take advantage of the Windows® environment. See [Introduction to IBM Standard EHLLAPI, IBM Enhanced EHLLAPI and WinHLLAPI Programming on page 7](#).
- Any 32-bit APIs which accept\return Window Handles and pointers might not work correctly with HCL ZIEWin due to difference in pointer\handle sizes between x86 and x64 platforms.

For Example:

"Data String" parameter returned in byte numbers (9-12) in API Start Communication Notification (80) might be truncated on x64 platform.

- **Z and I Emulator for Windows Session API (PCSAPI):** This interface is used to start, stop, and control emulator sessions and settings. See [PCSAPI Functions on page 188](#).

For Z and I Emulator for Windows Version 16.0, functions have been added to allow control and retrieval of page and printer settings. See [Page Setup Functions on page 197](#) and [Printer Setup Functions on page 205](#).

- **HCL Z and I Emulator for Windows Host Access Class Library (ECL):** ECL is a set of objects that allow application programmers and scripting language writers to access host applications easily and quickly. Z and I Emulator for Windows supports three different ECL layers (C++ objects, ActiveAutomation (OLE), and LotusScript Extension (LSX)). Refer to *Host Access Class Library (HACL)* for more details.

Using API Header Files

The application program should include operating system header files before including API header files. For example:

```
#include <windows.h>    // Windows main header
#include "pcsapi.h"      // ZIEWin PCSAPI header
...
```

Critical Sections

Use critical sections (**EnterCriticalSection** function) carefully when your program calls emulator APIs. Do not make emulator API calls within a critical section. If one thread of an application establishes a critical section and another thread is within an emulator API call, the call is suspended until you exit from the critical section.

During processing of an API call, all signals (except numeric coprocessor signals) are delayed until the call completes or until the call needs to wait for incoming data. Also, **TerminateProcess** issued from another process is held until the application completes an API call it might be processing.

Stack Size

Emulator APIs use the calling program's stack when they are executed. The operating system, the application, and the API all require stack space for dynamic variables and function parameters. At least 8196 bytes (8K) of stack space should be available at the time of an API call. It is the responsibility of the application program to ensure sufficient stack space is available for the API.

Windows x64 Platform Support

The x64-based versions of Microsoft® Windows® Server 2008 and Microsoft® Windows® 8/8.1/10 x64 Edition are optimized to run native 64-bit programs, but do not support 32-bit drivers or 16-bit applications.

For these platforms, Z and I Emulator for Windows does not install the following libraries.

- 16-bit API support:
 - Standard EHLLAPI 16-bit interface
 - WinHLLAPI 16-bit interface
 - PCSAPI 16-bit interface

Sample Programs

Several sample programs are provided, each of which illustrates the use of one of the Z and I Emulator for Windows APIs. If you choose to install the sample programs, they will be installed in the \SAMPLES directory.



Note: International Business Machines Corporation provides these files as is, without warranty of any kind, either express or implied, including, but not limited to, the implied warranties of merchantability or fitness for a particular purpose.

The sample program files include source and supporting files for the following Z and I Emulator for Windows APIs:

- Emulator High-Level Language Programming Interface (EHLLAPI)
- PCSAPI Functions

The following files are installed in the \SAMPLES directory.

Table 1. Sample Program Subdirectories

File Name	Description
DDE_C.H	DDE include file
EHLAPI32.H	IBM® standard 32-bit EHLLAPI include file
WHLLAPI.H	WinHLLAPI 16-bit include file
HAPI_C.H	EHLLAPI include file
PCSAPI.H	PCSAPI include file
PCSCALLS.LIB	Import library for standard interface
PCSCAL32.LIB	Import library for enhanced interface
EHLAPI32.LIB	Import library for IBM® Standard 32-bit EHLLAPI interface
WHLLAPI.LIB	Import library for WinHLLAPI 16-bit interface
WHLAPI32.LIB	Import library for WinHLLAPI 32-bit interface

The following subdirectories are created in the \SAMPLES directory.

Table 2. Sample Program Subdirectories

File Name	Description
ECL	The sample\ec\cpp folder contains all files related HACL CPP sample. The sample\ec\vb folder contains all files related to HACL VB.Net sample.
HLLSMP	Shows how to use EHLLAPI to request a keystroke and log on to a VM system.(X86). It supports the logon, pastetext and sendkey functionalities. Refer to the hllsmp\ Readme.txt for the details on using the above-mentioned functionalities with hllsmp.exe. (X64).

Chapter 2. Introduction to IBM Standard EHLLAPI, IBM Enhanced EHLLAPI and WinHLLAPI Programming

This chapter provides information needed to incorporate IBM® Standard EHLLAPI (16- and 32-bit), WinHLLAPI (16- and 32-bit), and IBM® Enhanced 32-bit EHLLAPI (EHLAPI32) functions into applications written in a high level language. It provides details on call format, memory allocation considerations, initializing the interfaces, and compiling and linking applications. Also included is a short sample EHLLAPI program and the compile/link instructions used to build it. Finally, a set of possible uses for the EHLLAPI interface (scenarios) is described.

An EHLLAPI application is any application program which uses the EHLLAPI interface to access the host 3270/5250/VT presentation space. The presentation space includes the visible emulator character data, fields and attribute data, keystroke data, and other information.

EHLLAPI Overviews

Following are overviews for HLLAPI programming interfaces.

IBM Standard EHLLAPI

EHLLAPI is a standard programming interface which allows programmatic access to a host emulator session. Functions are provided for reading host screen data (such as the characters and attributes), for sending keystrokes, and performing other emulator-related functions.

The EHLLAPI interface is a single call-point interface. There is a single callable API through which all EHLLAPI functions are requested. On each call to the interface the application provides a function number which identifies the function requested, a pointer to a data buffer, a pointer to the length of the data buffer, and a pointer to a return code (see [EHLLAPI Call Format on page 8](#)).

WinHLLAPI

WinHLLAPI is based on the familiar EHLLAPI API. It encompasses all of the existing functionality and adds extensions that take advantage of the Windows® message driven environment. Users of the HCL Z and I Emulator for Windows EHLLAPI interface will notice no functional difference unless they incorporate the WinHLLAPI extensions.

The WinHLLAPI extension functions and any functions that deviate from the EHLLAPI form are described in [WinHLLAPI Extension Functions on page 175](#). For information on common functions, refer to [EHLLAPI Functions on page 31](#).

WinHLLAPI and IBM® Standard EHLLAPI

The entry symbol for WinHLLAPI, is appropriately, **WinHLLAPI**. EHLLAPI users wishing to switch to the WinHLLAPI implementation must change from the **hllapi** standard entry. New users should follow all of the directions in [EHLLAPI Functions on page 31](#), and use the **WinHLLAPI** entry in place of the standard **hllapi** entry.

IBM Enhanced EHLLAPI and IBM Standard EHLLAPI

IBM Enhanced EHLLAPI is based on the familiar EHLLAPI API. It encompasses all of the existing functionality but takes advantage of the 32-bit environment and uses modified data structures. Standard interface users wishing to switch to IBM® Enhanced 32-bit EHLLAPI need to change only the entry symbol from LPWORD to LPINT in the first, third, and fourth parameters. New users should use the procedures in the following sections.

Languages

Any programming language which can invoke an entry point in a DLL with the "Pascal" calling convention can be used to execute EHLLAPI functions. However, the Z and I Emulator for Windows EHLLAPI toolkit provides header files and function prototypes only for the C++ languages. A clear understanding of data structure layout and calling conventions is required to use any other language. The EHLLAPI toolkit supports the following C/C++ compilers:

- Microsoft® Visual C/C++ Version 4.0 and higher

Most other C/C++ compilers will also work with the toolkit.

EHLLAPI C/C++ applications must include the Z and I Emulator for Windows EHLLAPI header file (HAPI_C.H). This file defines the layout of data structures and provides a prototype for the EHLLAPI entry point.



Note: The data structure layout for 16- and 32-bit applications are not the same (see [Standard and Enhanced Interface Considerations on page 25](#)).

EHLLAPI Call Format

The EHLLAPI entry point (**hllapi**) is always called with the following four parameters:

1. EHLLAPI Function Number (input)
2. Data Buffer (input/output)
3. Buffer Length (input/output)
4. Position (input); Return Code (output)

The prototype for IBM® Standard EHLLAPI is:

```
[long hllapi (LPWORD, LPSTR, LPWORD, LPWORD);
```

The prototype for IBM® Enhanced EHLLAPI is:

```
[long hllapi (LPINT, LPSTR, LPINT, LPINT);
```

Each parameter is passed by *reference* not by value. Thus each parameter to the function call must be a *pointer* to the value, not the value itself. For example, the following is a correct example of calling the EHLLAPI Query Session Status function:


```

#include "hapi_c.h"
struct HLDQuerySessionStatus QueryData;
int    Func, Len, Rc;
long   Rc;

memset(QueryData, 0, sizeof(QueryData)); // Init buffer
QueryData.qsst_shortcode = 'A';          // Session to query
Func = HA_QUERY_SESSION_STATUS;          // Function number
Len = sizeof(QueryData);                  // Len of buffer
Rc = 0;                                  // Unused on input

hllapi(&Func, (char *)&QueryData, &Len, &Rc); // Call EHLLAPI
if (Rc != 0) {                             // Check return code
    // ...Error handling
}

```

All the parameters in the **hllapi** call are pointers and the return code of the EHLLAPI function is returned in the value of the 4th parameter, not as the value of the function. For example, the following is **not** correct:

```

if (hllapi(&Func, (char *)&QueryData, &Len, &Rc) != 0) { // WRONG!
    // ...Error handling
}

```

Although the **hllapi** function is defined to return a **long** data type for IBM® Standard and Enhanced EHLLAPI, and **void** data type for WinHLLAPI, its value is undefined and should not be used.

The second through fourth parameters of the **hllapi** call can return information to the application. The description of each EHLLAPI function describes what, if any, information is returned in these parameters.

Data Structures

Many EHLLAPI functions use a formatted data structure to pass information to or from the application program. The description of each function shows the layout of the data structure. The data passed to or from the EHLLAPI function must exist in storage exactly as documented, byte for byte. Note that the structure layout is the same for all IBM® Standard and WinHLLAPI 16- and 32-bit applications. Data structures for the IBM® Enhanced 32-bit applications are packed to a 4-byte alignment.

It is *highly recommended* that the supplied header file and data structure definitions be used to ensure proper data alignment and layout. Although it is technically possible, the following is *not* recommended:

```

char QueryData[20]; // Not recommended
...
Func = HA_QUERY_SESSION_STATUS;
hllapi(&Func, QueryData, &Len, &Rc);
if (QueryData[13] == 'F') {
    // ...this is a 5250 session
}

```

The recommended way to write this function would be:

```

#include "hapi_c.h"
struct HLDQuerySessionStatus QueryData; // Recommended
...

```

```

Func = HA_QUERY_SESSION_STATUS;
hllapi(&Func, (char *)&QueryData, &Len, &Rc);
if (QueryData.qsst_sestype == 'F') {
    // ...this is a 5250 session
}

```

Memory Allocation

EHLAPI functions do not allocate or free memory. The application program must preallocate buffer space for EHLAPI functions which require it before calling the **hllapi** entry point. The buffer space may be pre-allocated as a dynamic variable such as:

```
struct HLDQuerySessionStatus QueryBuff;
```

or it may be allocated by a call to a C library or operating system function such as:

```

struct HLDQuerySessionStatus *QueryBuff;
...
QueryBuff = malloc(sizeof(struct HLDQuerySessionStatus));

```

In any case, the application is responsible for allocating sufficient buffer space before calling EHLAPI functions and for freeing buffers when they are not needed.

EHLAPI Return Codes

EHLAPI functions return a completion code or return code in the 4th parameter of the **hllapi** function call (except for the **Convert Position** or **RowCol** (99) function). The return code indicates the success or failure of the requested function.

Unless indicated otherwise in the description of each function, the following table shows the meaning of each return code value. Some functions may have a slightly different interpretation of these return codes; refer to the individual function descriptions for details.

Table 3. EHLAPI Return Codes

Return Code	Explanation
0	The function successfully executed, or no update since the last call was issued.
1	An incorrect host presentation space ID was specified. The specified session either was not connected, does not exist, or is a logical printer session.
2	A parameter error was encountered, or an incorrect function number was specified. (Refer to the individual function for details.)
4	The execution of the function was inhibited because the target presentation space was busy, in X CLOCK state (X []), or in X SYSTEM state.
5	The execution of the function was inhibited for some reason other than those stated in return code 4.
6	A data error was encountered due to specification of an incorrect parameter (for example, a length error causing truncation).

Table 3. EHLLAPI Return Codes (continued)

Return Code	Explanation
7	The specified presentation space position was not valid.
8	A functional procedure error was encountered (for example, use of conflicting functions or missing prerequisite functions).
9	A system error was encountered.
10	This function is not available for EHLLAPI.
11	This resource is not available.
12	This session stopped.
24	The string was not found, or the presentation space is unformatted.
25	Keystrokes were not available on input queue.
26	A host event occurred. See Query Host Update (24) for details.
27	File transfer was ended by a Ctrl+Break command.
28	Field length was 0.
31	Keystroke queue overflow. Keystrokes were lost.
32	An application has already connected to this session for communications.
33	Reserved.
34	The message sent to the host was canceled.
35	The message sent from the host was canceled.
36	Contact with the host was lost.
37	Inbound communication has been disabled.
38	The requested function has not completed its execution.
39	Another DDM session is already connected.
40	The disconnection attempt was successful, but there were asynchronous requests that had not been completed at the time of the disconnection.
41	The buffer you requested is being used by another application.
42	There are no outstanding requests that match.
43	The API was already locked by another EHLLAPI application (on LOCK) or API not locked (on UN-LOCK).

Compiling and Linking

The application program can be linked using dynamic linking method. This means that we can link the entry point by performing the dynamic linking. In this case, the application uses operating system calls to load the correct DLL and obtain the entry point address at run time.

The following table shows which .DLL should be used for dynamic loading.

Interface	Entry Point	DLL
IBM® Standard (64-bit)	hllapi	EHLAPI32.DLL
IBM® Enhanced (64-bit)	hllapi	PCSHLL32.DLL
WinHLLAPI (64-bit)	winhllapi	WHLAPI32.DLL

Dynamic Link Method

Using the dynamic link method the application makes calls to the operating system at run time to load the Z and I Emulator for Windows EHLLAPI module and to locate the **hllapi** entry point within it. This method requires more code in the application but gives the application greater control over error conditions. For example, the application can display a specific error message to the user if the Z and I Emulator for Windows EHLLAPI module cannot be found.

To use dynamic linking, the application needs to load the appropriate Z and I Emulator for Windows module and locate the entry point. It is recommended that the entry point be located by its ordinal number and not by name. The ordinal number is defined in the header file. The following 32-bit Windows® code loads the IBM® Standard 32-bit EHLLAPI module, locates the **hllapi** entry point, and makes an EHLLAPI function call.

```
#include "hapi_c.h"

HMODULE Hmod; // Handle of PCSHLL32.DLL
long (APIENTRY hllapi)(int *, char *, int *, int *); // Function pointer
int HFunc, HLen, HRc; // Function parameters
char HBuff[1]; // Function parameters

Hmod = LoadLibrary("PCSHLL32.DLL"); // Load EHLLAPI module
if (Hmod == NULL) {
    // ... Error, cannot load EHLLAPI module
}

hllapi = GetProcAddress(Hmod, MAKEINTRESOURCE(ord_hllapi));
// Get EHLLAPI entry point

if (hllapi == NULL) {
    // ... Error, cannot find EHLLAPI entry point
}

HFunc = HA_RESET_SYSTEM; // Run EHLLAPI function
HLen = 0;
HRc = 0;
(*hllapi)(&HFunc, HBuff, &HLen, &HRc);
if (HRc != 0) {
    // ... EHLLAPI access error
}
```

Multithreading

HCL Enhanced EHLLAPI (32-bit) and HCL® Standard EHLLAPI 16-bit connect on a per process basis. All threads access the same connected host session. The thread that performs the connections must also perform the disconnection.

HCL® Standard EHLLAPI (32-bit) and WinHLLAPI connect on a per thread basis. Each thread must maintain its own connections. This allows a multithreaded process to maintain connections to more than one connected host session at a time. This eliminates the need for multi-process schemes when using a WinHLLAPI program to coordinate data between different hosts. It also puts the burden of connecting and disconnecting as necessary on the individual thread.

Presentation Spaces

Many EHLLAPI functions require a *presentation space ID (PSID)* to indicate which host emulator session is to be used for the function. (This is also referred to as the *short session ID*). A presentation space ID is a single character in the range A to Z. There are a maximum of 26 sessions.

IBM® Enhanced 32-Bit Interface Presentation Space IDs

For IBM® Enhanced EHLLAPI applications, the session ID is extended with three additional bytes. These extended session bytes must be set to zero for future compatibility. This is most easily accomplished by setting the contents of EHLLAPI buffers to all binary zero before filling them in with the required information. For example, the following might be used to query the status of session B:

```
#include "hapi_c.h"
int HFunc, HLen, HRc;           // Function parameters
struct HLDPMWindowStatus StatusData; // Function parameters

Func = HA_PM_WINDOW_STATUS;
HLen = sizeof(StatusData);
HRc = 0;

// Set data buffer to zeros and fill in request
memset(&StatusData, 0x00, sizeof(StatusData));
StatusData.cwin_shortname = 'B'; // Short session ID
StatusData.cwin_option = 0x02; // Query command

hllapi(&Func, (char *)&StatusData, &HLen, &HRc);
```

Types of Presentation Spaces

An emulator session can be configured as a display session or a printer session. EHLLAPI applications cannot connect to printer or router sessions of PC400. The **Query Sessions (10)** function can be used to determine the type of a particular session.

Size of Presentation Spaces

An emulator display session can be configured for a range of screen sizes from 1920 bytes (24x80 screen size) to 9920 bytes (62x160 screen size). Some EHLLAPI functions such as **Copy PS to String (8)** require the application to allocate enough storage to hold (possibly) the entire presentation space. The size of the presentation space for a given session can be obtained using the **Query Session Status (22)** function.

Presentation Space IDs

EHLLAPI functions interact with only one presentation space at a time. The presentation space ID (PSID) is used to identify the particular presentation space in which a function is to operate.

For some functions, the PSID is contained in a preceding call to the **Connect Presentation Space** (1) function. For other functions, the PSID is contained in the calling data string parameter.

Host-Connected Presentation Space

Connection to the host presentation space (or session) is controlled by using the **Connect Presentation Space** (1) and **Disconnect Presentation Space** (2) functions. The status of the connection determines whether some functions can be executed. It also affects how the PSID is defined. The following text explains how to control the status of the connection to the host presentation space:

- At any given time, there can be either no host-connected presentation space, or there can be one and only one host-connected presentation space.
 - There is no default host-connected presentation space.
 - Following a connect, there is one and only one host-connected presentation space. The host presentation space that is connected is identified in the calling data string parameter of the connect function.
 - A subsequent call to connect can be executed with no intervening disconnect. In this case, there is still one and only one host-connected presentation space. Again, the host presentation space that is connected is identified in the calling data string parameter of the connect function.
 - Following a disconnect, there is no host-connected presentation space. This rule applies following multiple consecutive calls to connect or following a single call to connect.
 - You cannot connect to a logical printer session.
-

Presentation Space ID Handling

The PSID is used to specify the host presentation space (or session) in which you desire a function to operate. The way the PSID is handled is affected by two factors:

1. The method used to specify the PSID:
 - a. As the calling data string parameter of a preceding call to the **Connect Presentation Space** (1) function
 - b. As a character in the calling data string of the function being executed. Handling varies depending on whether the character is:
 - A letter A through Z
 - A blank or a null
2. The status of the connection to the host presentation space.

The following paragraphs describe how the PSID is handled for the various combinations of these two factors.

PSID Handling for Functions Requiring Connect

Some functions interact only with the host-connected presentation space. These functions require the **Connect Presentation Space** (1) function as a prerequisite call. The PSID for these functions is determined by the **Connect Presentation Space** (1) and the **Disconnect Presentation Space** (2) functions as follows:

- When there is no host-connected presentation space, these functions do not interact with any presentation space. A return code of 1 is generated.
 - When there is one host-connected presentation space, these functions interact with the presentation space specified in the calling data string parameter of the most recent call to the **Connect Presentation Space** (1) function.
-

PSID Handling for Functions Not Requiring Connect

Some functions can interact with a host presentation space whether it is connected or not. These functions allow you to specify the PSID in the calling data string parameter. They are as follows:

- **Connect Presentation Space** (1)
- **Convert Position RowCol** (99)
- **Get Key** (51)
- **Post Intercept Status** (52)
- **Query Close Intercept** (42)
- **Query Host Update** (24)
- **Query Session Status** (22)
- **Start Close Intercept** (41)
- **Start Host Notification** (23)
- **Start Keystroke Intercept** (50)
- **Stop Close Intercept** (43)
- **Stop Host Notification** (25)
- **Stop Keystroke Intercept** (53)

All except the first two of these functions allow you to specify the PSID using either:

- A letter A through Z
- A blank or a null

The first two functions require that a letter be used to specify the PSID.

When there is no host-connected presentation space, the following rules apply:

- The function can interact with any host presentation space if a letter, not a blank or a null, is used to specify the PSID.
- If a blank or a null is used to specify the PSID, a return code of 1 is generated. The function does not execute.
- Using a letter to specify the PSID does not establish a host-connected presentation space, except on a connect PS request.

When there is one host-connected presentation space, the following rules apply:

- The function can interact with any host presentation space if a letter is used to specify the PSID.
- If a blank or a null is used to specify the PSID, the function operates in the presentation space identified in the most recent call to the **Connect Presentation Space** (1) function.
- Using a letter to specify the PSID does not change the established PSID of the host-connected presentation space, except on a connect PS request.

The following functions are available for printer sessions:

- **Start Host Notification** (23)
- **Query Host Update** (24)
- **Stop Host Notification** (25)

Sharing EHLLAPI Presentation Space between Processes

More than one EHLLAPI application can share a presentation space if the applications support sharing (that is, if they were developed to work together or if they exhibit predictable behavior¹). To determine which applications support sharing, EHLLAPI applications are specified as one of following types:

- Supervisory
- Exclusive write with read privilege allowed
- Exclusive write without read privilege allowed
- Super write
- Read

The type of shared access can be defined by setting the following read and write sharing options for each function in the **Set Session Parameters** (9) function call:

SUPER_WRITE

The application allows other applications that allow sharing and have write access permissions to concurrently connect to the same presentation space. The originating application performs supervisory-type functions but does not create errors for other applications that share the presentation space.

1. This means that two EHLLAPI programs will not be vying for the same Presentation Space at the same time; or that there is logic in those programs which will allow the program to wait until the PS is available; or that the applications never use the Session in a way which would lock out other applications.

WRITE_SUPER

The application requires write access and allows only supervisory applications to concurrently connect to its presentation space. This is the default value.

WRITE_WRITE

The application requires write access and allows partner or other applications with predictable behavior to share the presentation space.

WRITE_READ

The application requires write access and allows other applications that perform read-only functions to share the presentation space. The application is also allowed to copy the presentation space and perform other read-only operations as usual.

WRITE_NONE

The application has exclusive use of the presentation space. No other applications are allowed to share the presentation space, including supervisory applications. The application is allowed to copy the presentation space and perform read-only operations as usual.

READ_WRITE

The application requires only read access to monitor the presentation space and allows other applications that perform read or write, or both, functions to share the presentation space. The application is also allowed to copy the presentation space and perform other read-only operations as usual.



Note: Sharing presentation space is not available between threads in a process.

Table 4. EHLLAPI Read and Write Sharing Option Combinations

Calling Application	Super_Write	Write_Super	Write_Write	Write_Read	Write_None	Read_Write
Super_Write	Yes	Yes	Yes	No	No	Yes
Write_Super (default)	Yes	No	No	No	No	No
Write_Write	Yes	No	Yes	No	No	Yes
Write_Read	No	No	No	No	No	Yes
Write_None	No	No	No	No	No	No
Read_Write	Yes	No	Yes	Yes	No	Yes

In addition to specifying compatible read and write access options, applications that are designed to work together but cannot allow others to work in the same presentation space can optionally define a keyword, KEY\$nnnnnnnn, in

the **Set Session Parameters** (9) function call. This keyword allows only those applications that use the same keyword to share the presentation space.

**Note:**

1. The **Start Keystroke Intercept** (50) function is non-shareable. Only one application at a time can trap keystrokes.
2. The **Connect To Presentation Space** (1) and **Start Keystroke Intercept** (50) functions share common subsystem functions. Successful requests by an application to share either of these functions can affect the requests of these two functions by other applications. For example, if application A successfully requests a **Connect To Presentation Space** (1) with Write_Read access and KEY \$abcdefgh as the keyword, a request by application B to **Connect To Presentation Space** (1) or **Start Keystroke Intercept** (50) is successful only if both applications have set compatible read and write options.

Table 5. Prerequisite Functions and Associated Dependent Functions

Prerequisite Call	Functions	Access
Allocate Communications Buffer (120)	Free Communication Buffer (120)	N/A
Connect Window Service (101)	Change PS Window Name (106) Change Switch List Name (105) Disconnect Window Service (102) Query Window Service (103) Window Status (104)	Write Read Query=Read Set=Write Write
Connect Presentation Space (1)	Copy Field to String (34) Copy OIA (13) Copy Presentation Space (5) Copy Presentation Space to String (8) Copy Presentation Space to Clipboard (35) Copy String to Field (33) Copy String to Presentation Space (15) Disconnect Presentation Space (2) Find Field Length (32) Find Field Position (31) Query Cursor Location (7) Query Field Attribute (14) Paste Clipboard to Presentation Space (36)	Read Read Read Read Read Write Write Write Read Read Read Read Write

Table 5. Prerequisite Functions and Associated Dependent Functions (continued)

Prerequisite Call	Functions	Access
	Release (12)	Write
	Reserve (11)	Write
	Search Field (30)	Read
	Search Presentation Space (6)	Read
	Send key (3)	Read
	Set Cursor (40)	Write
	Start Playing Macro (110)	Write
	Wait (4)	Read
Connect Structured Field (120)	Disconnect Structured Field (121) Get Request Completion (125) Read Structured Field (126) Write Structured Field (127)	N/A
Read Structured Field (126)	Get Request Completion (125)	N/A
Start Close Intercept (41)	Query Close Intercept (42) Stop Close Intercept (43)	N/A
Start Host Notification (23)	Query Host Update (24) Stop Host Notification (25)	
Start Keystroke Intercept (50)	Get Key (51) Post Intercept Status (52) Stop Keystroke Intercept (53) Send Key (3) if edit keystrokes are to be sent (edit keystroked support is available in Enhanced Mode)	N/A
Write Structured Field (127)	Get Request Completion (125)	N/A

Locking Presentation Space

An application, even if specified with shared presentation space, can obtain exclusive control of a presentation space by using the **Lock Presentation Space API** (60) or the **Lock Windows® Services API** (61) functions. Requests by the other applications to use a presentation space locked by these functions are queued and processed in first-in-first-out (FIFO) order when the originating application unlocks the presentation space.

If the application that locked the presentation space does not unlock it by using the same call with an **Unlock** option or **Reset System** (21) call, the lock is removed when the application terminates or the session stops.

Using mouse actions to select, copy, and paste text in the Presentation Space

The following mouse actions can be used in the Presentation Space.

- Select a word by double-clicking the left mouse button.
 - Copy a selected word by clicking the right mouse button.
 - Paste a copied word by double-clicking the mouse right button.
-

ASCII Mnemonics

Keystrokes originating at a host keyboard might have a corresponding ASCII value. The response of the **Get Key** (51) function to a keystroke depends on whether the key is defined and also on whether the key is defined as an ASCII value or an ASCII mnemonic.

The keyboard for one session might not be capable of producing some codes needed by the another session. ASCII mnemonics that represent these codes can be included in the data string parameter of the **Send Key** (3) function.

The capabilities of the **Send Key** (3) function and the **Get Key** (51) function allow sessions to exchange keystrokes that might not be represented by ASCII values or by an available key. A set of mnemonics that can be generated from a keyboard is provided. These mnemonics let you use ASCII characters to represent the special function keys of the workstation keyboard.

Mnemonics for unshifted keys consist of the escape character followed by an abbreviation. This is also true for the shift keys themselves, Upper shift, Alt, and Ctrl. Mnemonics for shifted keys consist of the mnemonic for the shift key followed by the mnemonic for the unshifted key. Hence the mnemonic for a shifted key is a 4-character sequence of escape character, abbreviation, escape character, abbreviation.

The default escape character is `@`. You can change the value of the escape character to any other character with the `ESC=c` option of the **Set Session Parameters** (9) function. The following text uses the default escape character, however.

Shift indicators that are not part of the ASCII character set are represented to the host application by 2-byte ASCII mnemonics as follows:

Upper shift	@S
Alt	@A
Ctrl	@r

Mnemonics for these shift indicators are never received separately by an application. Likewise, they are never sent separately by an application. Shift indicator mnemonics are always accompanied by a non-shift-indicator character or mnemonic.

The abbreviations used make the mnemonics for special keys easy to remember. An alphabetic key code has been used for the most common keys. For example, the Clear key is *C*; the Tab key is *T*, and so on. Please note that the uppercase and lowercase alphabetic characters are mnemonic abbreviations for different keys.

The following text describes the use of these functions.

General

All defined keys are represented by either:

- A 1-byte ASCII value that is part of the 256-element ASCII character set, or
- A 2-, 4-, or 6-byte ASCII mnemonic

To represent a key defined as an ASCII character, a 1-byte ASCII value that corresponds to that character is used.

To represent a key defined as a function, a 2-, 4-, or 6-byte ASCII mnemonic that corresponds to that function is used. For example, to represent the backtab key, `@B` is used. To represent PF1, `@1` is used. To represent Erase Input, `@A@F` is used. See the following lists:

<code>@B</code>	Left Tab	<code>@0</code>	Home	<code>@h</code>	PF17
<code>@C</code>	Clear	<code>@1</code>	PF1/F1	<code>@i</code>	PF18
<code>@D</code>	Delete	<code>@2</code>	PF2/F2	<code>@j</code>	PF19
<code>@E</code>	Enter	<code>@3</code>	PF3/F3	<code>@k</code>	PF20
<code>@F</code>	Erase EOF	<code>@4</code>	PF4/F4	<code>@l</code>	PF21
<code>@H</code>	Help (PC400)	<code>@5</code>	PF5/F5	<code>@m</code>	PF22
<code>@I</code>	Insert	<code>@6</code>	PF6/F6	<code>@n</code>	PF23
<code>@J</code>	Jump	<code>@7</code>	PF7/F7	<code>@o</code>	PF24
<code>@L</code>	Cursor Left	<code>@8</code>	PF8/F8	<code>@q</code>	End
<code>@N</code>	New Line	<code>@9</code>	PF9/F9	<code>@u</code>	Page UP (PC400)
<code>@O</code>	Space	<code>@a</code>	PF10/F10	<code>@v</code>	Page Down (PC400)
<code>@P</code>	Print	<code>@b</code>	PF11/F11	<code>@x</code>	PA1
<code>@R</code>	Reset	<code>@c</code>	PF12/F12	<code>@y</code>	PA2
<code>@T</code>	Right Tab	<code>@d</code>	PF13	<code>@z</code>	PA3
<code>@U</code>	Cursor Up	<code>@e</code>	PF14	<code>@@</code>	@ (at) symbol
<code>@V</code>	Cursor Down	<code>@f</code>	PF15	<code>@\$</code>	Alternate Cursor
<code>@Z</code>	Cursor Right				

<code>@A@C</code>	Test (PC400)	<code>@A@e</code>	Pink (PC/3270)
<code>@A@D</code>	Word Delete	<code>@A@f</code>	Green (PC/3270)
<code>@A@E</code>	Field Exit	<code>@A@g</code>	Yellow (PC/3270)
<code>@A@F</code>	Erase Input	<code>@A@h</code>	Blue (PC/3270)

@A@H	System Request	@A@i	Turquoise (PC/3270)
@A@I	Insert Toggle	@A@j	White (PC/3270)
@A@J	Cursor Select	@A@l	Reset Host Color (PC/3270)
@A@L	Cursor Left Fast	@A@t	Print (Personal Computer)
@A@Q	Attention	@A@u	Rollup (PC400)
@A@R	Device Cancel	@A@v	Rolldown (PC400)
@A@T	Print Presentation Space	@A@y	Forward Word Tab
@A@U	Cursor Up Fast	@A@z	Backward Word Tab
@A@V	Cursor Down Fast	@A@-	Field - (PC400)
@A@Z	Cursor Right Fast	@A@+	Field + (PC400)
@A@9	Reverse Video	@A@<	Record Backspace (PC400)
@A@b	Underscore (PC/3270)	@S@E	Print Presentation Space on Host (PC400)
@A@c	Reset Reverse Video (PC/3270)	@S@x	Dup
@A@d	Red (PC/3270)	@S@y	Field Mark

**Note:**

1. The first @ symbol in the first table represents the escape character. The first and second @ symbol in the second table is the escape character. The @ symbol is the default escape character. You can change the value of the escape character using the `ESC=c` option of the **Set Session Parameters (9)** function.

If you change the escape character to #, the literal sequences used to represent the Backtab, Home, and Erase Input keys become #B, #0, and #A#F, respectively.

2. If you send the mnemonic for print screen (that is, either @P or @A@T), place it at the end of the calling data string.
3. If you send the mnemonic for device cancel (that is, @A@R), it is passed through with no error message; however, local copy is not stopped.

Get Key (51) Function

If the terminal operator types a key defined as an ASCII character, the host application receives a 1-byte ASCII value that corresponds to that character.

If the operator types a key defined as a function, the host application receives a 2-, 4-, or 6-byte ASCII mnemonic that corresponds to that function. For example, if the **Backtab** key is typed, @B is received. If **PF1** is pressed, @1 is received. If **Erase Input** is pressed, @A@F is received.

If the operator types a defined shift key combination, the host application receives the ASCII character, or the 2-, 4-, or 6-byte ASCII mnemonic that corresponds to the defined character or function.

If the operator types an individual key that is not defined, the **Get Key** (51) function returns a return code of 20 and nothing is sent to the host application.

The **Get Key** (51) function prefixes all characters and mnemonics sent to the host application with two ASCII characters. The first ASCII character is the PSID of the host presentation space to which the keystrokes are sent. The other character is an A, S, or M for ASCII, special shift, or mnemonic, respectively. See [Return Parameters on page 86](#).

Send Key (3) Function

To send an ASCII character to another session, include that character in the data string parameter of the **Send Key** (3) function.

To send a function key to another session, include the ASCII mnemonic for that function in the data string parameter of the **Send Key** (3) function.

If the **Send Key** (3) function sends an unrecognized mnemonic to the host session a return code rejecting the key might result.

Debugging

As an aid in debugging EHLLAPI applications, the Trace Facility of Z and I Emulator for Windows may be used. This facility will produce a log of all EHLLAPI calls, parameters, return values, and return codes. For more information on using the Trace Facility, refer to *Administrator's Guide and Reference*.

A Simple EHLLAPI Sample Program

The following sample Windows® application will enter the character string "Hello World!" in the first input field of host session 'A'.

```
#include <stdlib.h>
#include <stdio.h>
#include <windows.h>
#include "hapi_c.h"

int main(char **argv, int argc) {
    int HFunc, HLen, HRc;
    char HBuff[1];
    struct HLDConnectPS ConnBuff;
    // Send Key string for HOME+string+ENTER:
    char SendString[] = "@0Hello World!@E";

    HFunc = HA_RESET_SYSTEM;
    HLen = 0;
    HRc = 0;
    hllapi(&HFunc, HBuff, &HLen, &HRc);
    if (HRc != HARC_SUCCESS) {
```

```

    printf("Unable to access EHLLAPI.\n");
    return 1;
}

HFunc = HA_CONNECT_PS;
HLen = sizeof(ConnBuff);
HRc = 0;
memset(&ConnBuff, 0x00, sizeof(ConnBuff));
ConnBuff.stps_shortcode = 'A';
hllapi(&HFunc, (char *)&ConnBuff, &HLen, &HRc);
switch (HRc) {
    case HARC_SUCCESS:
    case HARC_BUSY:
    case HARC_LOCKED: // All these are OK
        break;
    case HARC_INVALID_PS:
        printf("Host session A does not exist.\n");
        return 1;
    case HARC_UNAVAILABLE:
        printf("Host session A is in use by another EHLLAPI application.\n");
        return 1;
    case HARC_SYSTEM_ERROR:
        printf("System error connecting to session A.\n");
        return 1;
    default:
        printf("Error connecting to session A.\n");
        return 1;
}

HFunc = HA_SENDKEY;
HLen = strlen(SendString);
HRc = 0;
hllapi(&HFunc, SendString, &HLen, &HRc);
switch (HRc) {
    case HARC_SUCCESS:
        break;
    case HARC_BUSY:
    case HARC_LOCKED:
        printf("Send failed, host session locked or busy.\n");
        break;
    default:
        printf("Send failed.\n");
        break;
}

HFunc = HA_DISCONNECT_PS;
HLen = 0;
HRc = 0;
hllapi(&HFunc, HBuf, &HLen, &HRc);

printf("EHLLAPI program ended.\n");
return 0;
}

```

The application could be built with the following command:

```
nmake /a all
```


Standard and Enhanced Interface Considerations

There is no functional difference between the standard and enhanced EHLLAPI interfaces on a given platform.

However there are other important differences:

- The enhanced EHLLAPI interface extends the presentation space ID (PSID) from 1 byte to 4 bytes. Currently the additional bytes are not used, but your application should set them to binary zeros to ensure compatibility with future versions of enhanced EHLLAPI.
- The position (offset) of data elements in memory buffers passed to and from EHLLAPI functions are different. Data elements in enhanced EHLLAPI are aligned to double-word boundaries. Data elements in standard EHLLAPI are not aligned in any particular way. EHLLAPI applications should not be coded to set or retrieve data in the buffers by offset (byte) values. Instead, the supplied data structures in the HAPI_C.H file should be used to set and retrieve data elements. This will ensure that data is set and retrieved from the correct position for both 16- and 32-bit programs.

By prefilling EHLLAPI data buffers with binary zeros, and using the data structures supplied in HAPI_C.H, an application can be compiled for standard or enhanced operation without any source code changes. For example, the following section of code would work for standard EHLLAPI but would fail for enhanced EHLLAPI:

```
#include "hapi_c.h"
...
int Func, Len, Rc;
char Buff[18];
char SessType;

Func = HA_QUERY_SESSION_STATUS; // Function
Len = 18;                       // Buffer length
Rc = 0;
Buff[0] = 'A'                   // Session to query
hllapi(&Func, Buff, &Len, &Rc); // Execute function

SessType = Buff[9];              // Get session type
...
```

The above example would fail if compiled as an enhanced EHLLAPI application because:

- The application does not set the extended session ID bytes to zero.
- The buffer length for this function is 20, not 18.
- The session type indicator is not at offset 9 in the data buffer, it is at offset 12.

The following is the same function written to work correctly if compiled for standard or enhanced operation. Changed lines are indicated with a >:

```
#include "hapi_c.h"
...
int Func, Len, Rc;
> struct HLDQuerySessionStatus Buff;
char SessType;
```

```

Func = HA_QUERY_SESSION_STATUS;    // Function
> Len = sizeof(Buff);              // Buffer length
Rc = 0;
> memset(&Buff, 0x00, sizeof(Buff)); // Zero buffer
> Buff.qsst_shortcode = 'A';        // Session to query
hllapi(&Func, (char *)&Buff, &Len, &Rc); // Execute function

> SessType = Buff.qsst_sesstype;     // Get session type
...

```

Host Automation Scenarios

The sample scenarios presented here provide conceptual information about activities that can be facilitated by using EHLLAPI. The scenarios deal with the duties your EHLLAPI programmed operator can perform in these areas:

- Host system operation, including:
 - Search function
 - Sending keystrokes
- Distributed processing, including:
 - Data extraction
 - File transfer
- Integrating interfaces

Scenario 1. A Search Function

There are four phases in a typical host system transaction:

1. Starting the transaction
2. Waiting for the host system to respond
3. Analyzing the response to see if it is the expected response
4. Extracting and using the data from the response

Your programmed operator can use a series of EHLLAPI functions to mimic these actions. After determining the correct starting point for the host system transaction, the programmed operator can call the **Search Presentation Space** (6) function to determine which keyword messages or prompting messages are on the display screen.

Next, the programmed operator can use the **Send Key** (3) function to type data into a host system session and enter a host system transaction. Then the programmed operator can:

- Use the **Wait** (4) function that waits for the X CLOCK, X [], or X SYSTEM condition to end (or returns a keyboard-locked condition if the terminal has locked up).

If the keyboard is inhibited, your EHLLAPI program can call the **Copy OIA** (13) function to get more information about the error condition.

- Use the **Search Presentation Space** (6) function to look for an expected keyword to validate that the proper response had been received.
- Use the **Copy Presentation Space to String** (8) function (or any of several data access functions) to extract the desired data.

The **Search Presentation Space** (6) function is critical to simulate another task of the terminal operator. Some host systems do not stay locked in X CLOCK, X [], or X SYSTEM mode until they respond; instead, they quickly unlock the keyboard and allow the operator to stack other requests. In this environment, the terminal operator depends on some other visual prompt to know that the data has returned (perhaps a screen title or label). The **Search Presentation Space** (6) function allows your EHLLAPI program to search the presentation space while waiting. Also, while waiting for a response, calling the **Pause** (18) function allows other DOS sessions to share the central processing unit resource. The **Pause** (18) function has an option that allows your EHLLAPI program to wait for a host system update event to occur.

If no host system event occurs after a reasonable time-out period, your EHLLAPI program could call a customized error message such as:

```
No Response From Host.  Retry?
```

In this environment, program revisions become very important considerations, because the programmed operator must be reprogrammed for even minor changes in the display messages.

For example, if a terminal operator expects the message:

```
Enter Part Number:
```

as a prompt, he or she will probably be able to respond properly to an application change that produces the message:

```
Enter Component Number:
```

However, because the programmed operator is looking for a literal keyword string, subtle changes in message syntax, even as trivial as uppercase versus lowercase, can make the program take a preprogrammed error action.

Scenario 2. Sending Keystrokes

There are several considerations that demand attention in designing programs that send keystrokes to the host system. In some application environments, issuing a command is as simple as typing a string and pressing Enter. Other applications involve more complex formatted screens in which data can be entered into any one of several fields. In this environment you must understand the keystrokes required to fill in the display screen.

The Tab key mnemonic (@T; see [General on page 21](#) for a full list of mnemonics) can be used to skip between fields. When sending keystrokes to a field using the **Send Key** (3) function, you should be aware of the field lengths and contents. If you fill the fields completely and the next attribute byte is autoskip, your cursor will then be moved to the next field. If you then issued a tab, you would skip to yet another field.

Likewise, if your keystrokes do not completely fill the field, there might be data left from prior input. You should use the Erase End of Field (EOF) command to clear this residual data.

Scenario 3. Distributed Processing

Some applications fall into the category called *collaborative*. These applications provide a single end-user interface, but their processing is performed at two or more different physical locations.

An EHLLAPI application can interact with host system applications by intercepting the communication between the host system and the terminal user. The host system presentation space is the vehicle used to intercept this data. The local application can request to be notified each time the presentation space is updated or whenever an AID key is pressed by the operator.

This workstation application can then cooperate with a host system application in any of the following ways:

- On a field or presentation space basis using either the copy functions that address fields (**Copy String to Field** (33) function or **Copy Field to String** (34) function) or the functions that let you copy from and into presentation spaces (for example, **Copy String to Presentation Space** (15) function or **Copy Presentation Space to String** (8) function).
 - On a keystroke basis, using the **Send Key** (3) function.
 - On a file basis, for large blocks of data. You can have your application use the EHLLAPI file transfer capability (using **Send File** (90) function or **Receive File** (91) function) to transfer data or functions (such as load modules) and have it processed locally or remotely.
-

Scenario 4. File Transfer

In this scenario, assume that you want to automate a file transfer:

- You could begin by using the procedure discussed in the search scenario earlier to log on to a host system session.
 - Instead of using one of the copy functions (which are inefficient for copying many screens of data), your EHLLAPI program could call file transfer functions **Send File** (90) and **Receive File** (91) to transfer data.
 - Upon successful completion:
 - If the **Send File** (90) function finished executing, your EHLLAPI program could submit a batch job using either a copy function or the **Send Key** (3) function before logging off.
 - If the **Receive File** (91) function finished executing, your EHLLAPI program could start up a local application.
-

Scenario 5. Automation

An application can provide all the keystrokes for another application or can intersperse keystrokes to the target destination with those from the keyboard. Sometimes, to do this, the application must lock out other sources of keystroke input that might be destined for a target application or presentation space (using the **Reserve** (11) function) and the later unlock it (using the **Release** (12) function).

The origin of keystrokes presented to any application is determined by the design of the application. Keystrokes can originate from:

- The keyboard
- Data integrated into the source application
- Secondary storage retrieved through the DOS interface
- The Z and I Emulator for Windows interface

In all cases the keystrokes that are provided to the target application are indistinguishable from the ordinary operator input.

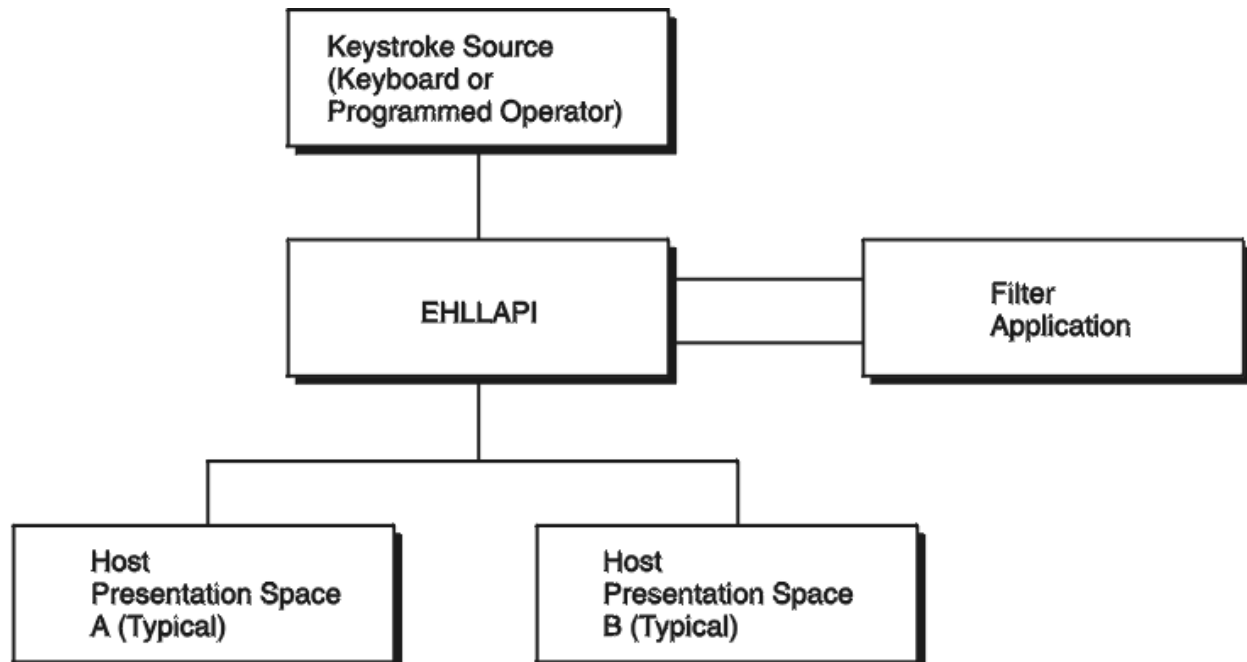
Scenario 6. Keystroke Filtering

An application that acts as a filter can intercept a keystroke coming from EHLLAPI (either from the keyboard or a source application) that is targeted for another destination. The keystroke can then be:

- Ignored (that is, deleted)
- Redirected to another application
- Validated
- Converted (for example, uppercase to lowercase)
- Enhanced (through keyboard macros)

[Figure 1: Keystroke Flow on page 29](#) provides a simplified representation of the keystroke flow and the objects within a keyboard enhancement environment.

Figure 1. Keystroke Flow



Scenario 7. Keyboard Enhancement

This scenario makes use of filtering to create an **enhancer application program**. An enhancer application program is one that monitors the data coming in from the keyboard and changes it in some specified way. Typically, these application programs use instructions called **keyboard macros**, which tell them what keystrokes to look for and what changes to make. The change might involve suppressing a keystroke (so it appears to the target application as though it was never sent), replacing a keystroke with another, or replacing single keystroke with a series of keystrokes.

To do this using EHLLAPI, you might construct this scenario:

1. Your EHLLAPI application program calls the **Connect Presentation Space** (1) function to connect to the presentation space whose keystrokes are to be filtered.
2. Your EHLLAPI program next calls the **Start Keystroke Intercept** (50) function specifying the L option. This causes all keystrokes to be routed to the filtering application program.
3. The filtering application program can now define a loop in which:
 - a. The **Get Key** (51) function intercepts all keystrokes being sent to the target presentation space.
 - b. The filtering application examines each keystroke and performs a keyboard macro task, such as:
 - Abbreviating program commands so that three- or four-keystroke command can be condensed into a single keystroke
 - Customizing commands so that they are easier to remember or consistent with other software packages
 - Creating **boiler plates** for contracts or frequently used letters
 - Rearranging the keyboard for concurrent applications that use the same keys for differing functions
 - c. For example, the filtering application might convert a key combination such as Alt+Y into a command to move the cursor to column 35 of the second line in presentation space and write the string "XYZ Tool Corporation, Dallas, Texas".
 - d. If a keystroke is rejected, your EHLLAPI program can cause a beep to be sounded, using the **Post Intercept Status** (52) function.
4. After your EHLLAPI program exits the filtering loop, **Stop Keystroke Intercept** (53) function to end the filtering process.

Chapter 3. EHLLAPI Functions

This chapter describes each individual Z and I Emulator for Windows EHLLAPI function in detail and explains how to use the EHLLAPI program sampler. The functions are arranged alphabetically by name. The functions are explained for both the standard and enhanced interfaces.



Note: Throughout this chapter WinHLLAPI, IBM® Standard 32-bit HLLAPI and 16-bit EHLLAPI are referred to as Standard Interface, and IBM® Enhanced 32-bit EHLLAPI is referred to as Enhanced Interface.

Page Layout Conventions

All EHLLAPI function calls are presented in the same format so that you can quickly retrieve the information you need. The format is:

- Function Name (Function Number)
 - Prerequisite Calls
 - Call Parameters
 - Return Parameters
 - Notes on Using This Function

Prerequisite Calls

“Prerequisite Calls” lists any calls that must be made prior to calling the function being discussed.

Call Parameters

“Call Parameters” lists the parameters that must be defined in your program to call the discussed EHLLAPI function and explains how those parameters are to be defined. If a parameter is never used by a function, then *NA* (not applicable) is listed. If a parameter can be overridden by certain values of session parameters defined with calls to the **Set Session Parameters** (9) function, such session parameters are named.

Return Parameters

“Return Parameters” lists the parameters that must be received by your program after a call to the discussed EHLLAPI function and explains how to interpret those parameters.

Notes on Using This Function

“Notes on Using This Function” lists any session options that affect the function under discussion. It also provides technical information about using the function and application development tips.

Summary of EHLLAPI Functions

Table 6: EHLLAPI Functions Summary on page 32 is the summary of the EHLLAPI functions:

Table 6. EHLLAPI Functions Summary

Function	3270	5250	VT
Connect Presentation Space (1) on page 40	Yes	Yes	Yes
Disconnect Presentation Space (2) on page 76	Yes	Yes	Yes
Send Key (3) on page 130	Yes	Yes	Yes
Wait (4) on page 163	Yes	Yes	Yes
Copy Presentation Space (5) on page 59	Yes	Yes	Yes
Search Presentation Space (6) on page 126	Yes	Yes	Yes
Query Cursor Location (7) on page 103	Yes	Yes	Yes
Copy Presentation Space to String (8) on page 64	Yes	Yes	Yes
Set Session Parameters (9) on page 139	Yes	Yes	Yes
Query Sessions (10) on page 109	Yes	Yes	Yes
Reserve (11) on page 122	Yes	Yes	Yes
Release (12) on page 121	Yes	Yes	Yes
Copy OIA (13) on page 49	Yes	Yes	Yes
Query Field Attribute (14) on page 103	Yes	Yes	Yes
Copy String to Presentation Space (15) on page 70	Yes	Yes	Yes
Pause (18) on page 95	Yes	Yes	Yes
Query System (20) on page 111	Yes	Yes	Yes
Reset System (21) on page 123	Yes	Yes	Yes
Query Session Status (22) on page 107	Yes	Yes	Yes
Start Host Notification (23) on page 153	Yes	Yes	Yes
Query Host Update (24) on page 106	Yes	Yes	Yes
Stop Host Notification (25) on page 162	Yes	Yes	Yes
Search Field (30) on page 124	Yes	Yes	Yes
Find Field Position (31) on page 82	Yes	Yes	Yes
Find Field Length (32) on page 80	Yes	Yes	Yes
Copy String to Field (33) on page 68	Yes	Yes	Yes
Copy Field to String (34) on page 45	Yes	Yes	Yes
Copy Presentation Space to Clipboard (35) on page 72	Yes	Yes	Yes
Paste Clipboard to Presentation Space (36) on page 74	Yes	Yes	Yes
Set Cursor (40) on page 138	Yes	Yes	Yes
Start Close Intercept (41) on page 149	Yes	Yes	Yes
Query Close Intercept (42) on page 99	Yes	Yes	Yes
Stop Close Intercept (43) on page 160	Yes	Yes	Yes

Table 6. EHLLAPI Functions Summary (continued)

Function	3270	5250	VT
Query Additional Field Attribute DRB on page 98	No	Yes	No
Start Keystroke Intercept (50) on page 156	Yes	Yes	Yes
Get Key (51) on page 85	Yes	Yes	Yes
Post Intercept Status (52) on page 97	Yes	Yes	Yes
Stop Keystroke Intercept (53) on page 163	Yes	Yes	Yes
Lock Presentation Space API (60) on page 91	Yes	No	No
Lock Window Services API (61) on page 93	Yes	No	No
Start Communication Notification (80) on page 151	Yes	Yes	Yes
Query Communication Event (81) on page 102	Yes	Yes	Yes
Stop Communication Notification (82) on page 161	Yes	Yes	Yes
Send File (90) on page 180	Yes	Yes	No
Receive File (91) on page 120	Yes	Yes	No
Cancel File Transfer (92) on page 35	Yes	Yes	Yes
Convert Position or Convert RowCol (99) on page 43	Yes	Yes	Yes
Connect Window Services (101) on page 42	Yes	Yes	Yes
Disconnect Window Service (102) on page 77	Yes	Yes	Yes
Query Window Coordinates (103) on page 113	Yes	Yes	Yes
Window Status (104) on page 164	Yes	Yes	Yes
Change Switch List LT Name (105) on page 37	Yes	Yes	Yes
Change PS Window Name (106) on page 36	Yes	Yes	Yes
Start Playing Macro (110) on page 159	Yes	Yes	Yes
Connect for Structured Fields (120) on page 38	Yes	No	No
Disconnect from Structured Fields (121) on page 75	Yes	No	No
Query Communications Buffer Size (122) on page 100	Yes	No	No
Allocate Communications Buffer (123) on page 33	Yes	No	No
Free Communications Buffer (124) on page 84	Yes	No	No
Get Request Completion (125) on page 88	Yes	No	No
Read Structured Fields (126) on page 115	Yes	No	No
Write Structured Fields (127) on page 169	Yes	No	No

Allocate Communications Buffer (123)

3270	5250	VT
Yes	No	No

The **Allocate Communications Buffer** function obtains a buffer from the operating system. A buffer address must be passed on both the **Read Structured Fields** (126) and **Write Structured Fields** (127) functions.

Prerequisite Calls

There are no prerequisite calls for this function.

Call Parameters

	Standard Interface	Enhanced Interface
Function Number	Must be 123	
Data String	See the following table	
Length	Must be 6	Must be 8
PS Position	NA	

The calling data string can contain:

Byte		Definition
Standard	Enhanced	
1–2	1–4	32-bit or 16-bit buffer length. (0 < size ≤ (64 KB-256 bytes)=X'FF00')
3–6	5–8	32-bit allocated buffer address (returned)

Return Parameters

Return Code	Explanation
0	The Allocate Communications Buffer function was successful.
2	An error was made in specifying parameters.
9	A system error occurred.
11	Resource unavailable (memory unavailable).

Notes on Using This Function

1. The EHLLAPI obtains a buffer from the operating system memory management and places the buffer address into the return parameter string. The requested buffer size (length) is also passed in the parameter string. The buffer size can be from 1 byte to 64 KB minus 256 bytes (X'FF00' bytes) in length.

See "**Query Communications Buffer Size (122)**" for information regarding buffer size.

2. Buffers obtained using this function must not be shared among different processes. If this is attempted, the applications will experience unpredictable results.
3. An EHLLAPI application must issue a **Free Communications Buffer (124)** function to free the allocated memory.
4. A maximum of 10 buffers can be allocated to an application. If this limit is reached, a return code for resource unavailable (RC=11) will be returned.
5. The **Reset System (21)** function frees buffers allocated by this function.

Cancel File Transfer (92)

3270	5250	VT
Yes	Yes	Yes

The **Cancel File Transfer** function causes any current EHLLAPI initiated **Send File** or **Receive File** for the specified session to immediately return.

Prerequisite Calls

Send File (90) or **Receive File** (91)

Call Parameters

	Enhanced Interface
Function Number	Must be 92
Data String	1-character short name of the host presentation space. A blank or null indicates request for updates to the host-connected presentation space
Length	4 is implied
PS Position	NA

The calling data structure contains these elements

Byte	Definition
1	A 1-character presentation space short name (PSID)
2-4	Reserved

Return Parameters

Return Code	Definition
0	The function was successful
1	An incorrect PSID was specified
8	No prior call to Start Communication Notification (80) function was called for the PSID
9	A system error was encountered

Notes on Using This Function

Since both **Send File** (90) and **Receive File** (91) are blocking calls, this function must always be issued on a different thread.

Change PS Window Name (106)

3270	5250	VT
Yes	Yes	Yes

The **Change PS Window Name** function allows the application to specify a new name for the presentation space window or reset the presentation space window to the default name.

Prerequisite Calls

Connect Window Services (101)

Call Parameters

	Standard Interface	Enhanced Interface
Function Number	Must be 106	
Data String	See the following table	
Length	Must be specified (See note.)	Must be 68
PS Position	NA	



Note: The data string length must be specified (normally 3–63 for PC/3270, 4–63 for PC400, 68 for enhanced interface).

The calling data string can contain:

Byte		Definition
Standard	Enhanced	
1	1	A 1-character presentation space short name (PSID)
	2–4	Reserved
2	5	A change request option value, select one of: <ul style="list-style-type: none"> • X'01' for changing the presentation space window name. • X'02' for resetting the presentation space window name.
3–63	6–66	An ASCII string of from 1 (for PC/3270) or 2 (for PC400) to 61 bytes including a terminator byte. The ASCII string must end with a NULL character. This string must contain at least one non-NULL character followed by a NULL character.
	67–68	Reserved

Return Parameters

Return Code	Explanation
0	The Change PS Window Name function was successful.
1	An incorrect host presentation space short session ID was specified, or the host presentation space was not connected.
2	An error was made in specifying parameters.
9	A system error occurred.
12	The session stopped.

Notes on Using This Function

A string is ended at the first NULL character found. The NULL character overrides the specified string length. If the NULL character is not at the end of the specified length, the last byte at the specified length is replaced by a NULL character, and the remainder of the data string is lost. If the NULL character is found before the specified length, the string is truncated at that point, and the remainder of the data string is lost.

If the application fails to reset the presentation space name before exiting, the exit list processing resets the name.

Change Switch List LT Name (105)

3270	5250	VT
Yes	Yes	Yes

The **Change Switch List LT Name** function allows the application to change or reset a switch list for a selected logical terminal (LT). The application must specify on the call the name to be inserted in the switch list.



Note: This is for compatibility with Communication Manager EHLLAPI, and has the same result as the **Change PS Window Name** (106) function.

Prerequisite Calls

Connect Window Services (101)

Call Parameters

	Standard Interface	Enhanced Interface
Function Number	Must be 105	
Data String	See the following table	
Length	Normally 4–63	Must be 68
PS Position	NA	

The calling data string can contain:

Byte		Definition
Standard	Enhanced	
1	1	A 1-character presentation space short name (PSID)
	2–4	Reserved
2	5	A change request option; select: <ul style="list-style-type: none"> • X'01' for changing a switch list LT name • X'02' for resetting a switch list LT name
3–63	6–66	An ASCII string of 2 to 61 bytes including a terminator byte. The ASCII string must end with a NULL character. This string must contain at least one non-NULL character followed by a NULL character.
	67–68	Reserved

Return Parameters

Return Code	Explanation
0	The Change Switch List LT Name function was successful.
1	An incorrect host presentation space short session ID was specified, or the host presentation space was not connected.
2	An error was made in specifying parameters.
9	A system error occurred.
12	The session stopped.

Notes on Using This Function

A string is ended at the first NULL character found. The NULL character overrides the specified string length. If the NULL character is not at the end of the specified length, the last byte at the specified length is replaced by a NULL character, and the remainder of the data string is lost. If the NULL character is found before the specified length, the string is truncated at that point, and the remainder of the data string is lost.

If the application fails to reset the switch list LT name before exiting, the exit list processing resets the name.

Connect for Structured Fields (120)

3270	5250	VT
Yes	No	No

The **Connect for Structured Fields** function allows an application to establish a connection to the emulation program to exchange structured field data with a host application. The workstation application must provide the Query Reply

data field and must point to it with in the parameter string. The destination/origin ID returned by the emulator will be returned to the application.

Prerequisite Calls

There are no prerequisite calls for this function.

Call Parameters

	Standard Interface	Enhanced Interface
Function Number	Must be 120	
Data String	See the following table	
Length	7 or 11	Must be 16
PS Position	NA	

The calling data string can contain:

Byte		Definition
Standard	Enhanced	
1	1	A 1-character presentation space short name (PSID)
	2–4	Reserved
2–5	5–8	Address of the Query Reply data buffer
6–7	9–10	Destination/origin unique ID. (16-bit word, returned)
	11–12	Reserved
8–11	13–16	The data in these position is ignored by EHLLAPI. However, no error is caused if the migrating program has data in these positions. This data is accepted to provide compatibility with migrating applications.

Return Parameters

Return Code	Explanation
0	The Connect for Structured Fields function was successful.
1	A specified host presentation space short session ID was not valid, or the host presentation space was not connected.
2	An error was made in specifying parameters.
9	A system error occurred.
10	The function is not supported by the emulation program.
32	An application has already connected to this session for communications (successful connect).
39	One DDM session is already connected to this session.

Notes on Using This Function

1. EHLLAPI scans the query reply buffers for the destination/origin ID (DOID) self-defining parameter (SDP) to determine the contents of the DOID field of the query reply. If this value is X'0000', the emulator will assign a DOID to the application and EHLLAPI will fill in the DOID field of the query reply with the assigned ID. If the value specified by the application in the DOID field of the query reply is a nonzero value, the emulator will assign the specified value as the application's DOID, assuming that the ID has not been previously assigned. If the specified DOID is already in use, a return code of 2 will be returned by EHLLAPI.
2. The application should build the Query Reply Data structures in the application's private memory. Refer to [Query Reply Data Structures Supported by EHLLAPI on page 220](#), for the detailed formats and usages of the query reply data structures supported by EHLLAPI.
3. Only cursory checking is performed on the Query Reply Data. Only the ID and the length of the structure are checked for validity.
4. Only one DDM base type connect is allowed per host session. If the DDM connection supports the self-defining parameter (SDP) for the destination origin ID (DOID), then multiple connects are allowed.
5. If return code RC=32 or RC=39 is received, an application is already connected to the selected session and use of that presentation space should be approached with caution. Conflicts with file transfer, and other EHLLAPI applications might result.

Connect Presentation Space (1)

3270	5250	VT
Yes	Yes	Yes

The **Connect Presentation Space** function establishes a connection between your EHLLAPI application program and the host presentation space.

Prerequisite Calls

There are no prerequisite calls for this function.

Call Parameters

	Standard Interface	Enhanced Interface
Function Number	Must be 1	
Data String	1-character short name of the host presentation space	
Length	1 is implied	Must be 4
PS Position	NA	

The calling data string can contain:

Byte		Definition
Standard	Enhanced	
1	1	A 1-character presentation space short name (PSID)
	2–4	Reserved

Return Parameters

The **Connect Presentation Space** function sets the return code to indicate the status of the attempt and, if successful, the status of the host presentation space.

Return Code	Explanation
0	The Connect Presentation Space function was successful; the host presentation space is unlocked and ready for input.
1	An incorrect host presentation space ID was specified. The specified session either does not exist or is a logical printer session. This return code could also mean that the API Setting for EHLLAPI is not set on.
4	Successful connection was achieved, but the host presentation space is busy.
5	Successful connection was achieved, but the host presentation space is locked (input inhibited).
9	A system error was encountered.
11	This resource is unavailable. The host presentation space is already being used by another system function.

Notes on Using This Function

1. The **Connect Presentation Space** function is affected by the `CONLOG/CONPHYS` session option.
2. An EHLLAPI application cannot be connected to multiple presentation spaces concurrently. Calls requiring the **Connect Presentation Space** function as a prerequisite use the currently connected presentation space. For example, if an application is connected to presentation space A, B, and C in that order, the application must connect to B or A again to issue functions.
3. Each thread that requests a **Connect Presentation Space** must have a corresponding **Disconnect Presentation Space** (2), or one of the threads must issue a **Reset System** (21), which affects all threads and disconnects any remaining connections.
4. More than one EHLLAPI application can share a presentation space, if the applications support sharing (that is, if they were developed to work together and if they exhibit predictable behavior) and have compatible read/write access and keyword options as set in the **Set Sessions Parameters** (9) function. For more information, see [Set Session Parameters \(9\) on page 139](#).
5. Because the **Connect Presentation Space** and **Start Keystroke Intercept** (50) functions share common subsystem functions, successful requests by an application to share either of these functions for the same session can affect the request of these two functions by other applications. For example, if application A successfully requests a **Connect Presentation Space** for a session with Write_Read access and KEY

\$abcdefg as the keyword, a request by application B to **Connect Presentation Space** for a session and **Start Keystroke Intercept** is successful only if both applications have set compatible read/write options.

6. You cannot connect to a session that is defined as a logical printer session. Refer to *Administrator's Guide and Reference* for more information.

Connect Window Services (101)

3270	5250	VT
Yes	Yes	Yes

The **Connect Window Services** function allows the application to manage the presentation space windows. Only one EHLLAPI application at a time can be connected to a presentation space for window services.

An EHLLAPI application can connect to more than one presentation space concurrently for window services.

Prerequisite Calls

There are no prerequisite calls for this function.

Call Parameters

	Standard Interface	Enhanced Interface
Function Number	Must be 101	
Data String	1-character short session ID of the host presentation space	
Length	1 is implied	Must be 4
PS Position	NA	

The calling data string can contain:

Byte		Definition
Standard	Enhanced	
1	1	A 1-character presentation space short name (PSID)
	2-4	Reserved

Return Parameters

Return Code	Explanation
0	The Connect Window Services function was successful.
1	An incorrect host presentation space short session ID was specified, or the Sessions Window Services manager was not connected. This return code could also mean that the API Setting for EHLLAPI is not set on.
9	A system error occurred.
10	The function is not supported by the emulation program.

Return Code	Explanation
11	This resource is unavailable. The host presentation space is already being used by another system function.

Notes on Using This Function

1. An EHLLAPI application can be connected to multiple presentation space windows at the same time. The application can go back and forth between the connected presentation space windows without having to disconnect. For example, if an application is connected to presentation space windows A, B, and C, the application can access all of A, B, and C at the same time, and the other applications cannot access A, B, or C.
2. A **Connect Window Services** function is sufficient for the process. However, each thread that requests a **Connect Window Services** must have a corresponding **Disconnect Window Services** (102), or one of the threads must issue a **Reset System** (21), which affects all threads and disconnects any remaining connections.

Convert Position or Convert RowCol (99)

3270	5250	VT
Yes	Yes	Yes

The **Convert Position** or **Convert RowCol** function converts the host presentation space positional value into the display row and column coordinates or converts the display row and column coordinates into the host presentation space positional value. This function does not change the cursor position.

Prerequisite Calls

There are no prerequisite calls for this function.

Call Parameters

	Standard Interface	Enhanced Interface
Function Number	Must be 99	
Data String	Host presentation space short name and P for the Convert Position function (for example, AP converts the presentation space position of session A); or Host presentation space short name and R for the Convert RowCol function (for example, AR converts the row and column coordinates of session A).	
Length	Row, when R is specified as the second character in the data string parameter. The lower limit for valid input is 1. The upper limit for valid input depends on how your host presentation space is configured. See Notes on Using This Function on page 45 . NA when P is specified as the second character in the data string parameter.	

	Standard Interface	Enhanced Interface
PS Position	Column, when R is specified as the second character in the data string parameter. The lower limit for valid input is 1. The upper limit for valid input ranges from 24 to 43 depending on how your host presentation space is configured. See Notes on Using This Function on page 45 .	Host presentation space position, when P is specified as the second character in the data string parameter. The lower limit for valid input is 1. The upper limit for valid input ranges from 1920 to 3564 depending on how your host presentation space is configured. See Notes on Using This Function on page 45 .

The calling data string can contain:

Byte		Definition
Standard	Enhanced	
1	1	A 1-character presentation space short name (PSID)
	2–4	Reserved
2	5	Convert option P or R
	6–8	Reserved

Return Parameters

This function returns a length and a return code.

Length:

For the **Convert Position** function (**P** as the second character in the calling data string), a number between 1 and 43 (for PC/3270) or 27 (for PC400) is returned. This value is the number of the row that contains the PS position contained in the calling PS position parameter. The upper limit can be smaller than 43 (for PC/3270) or 27 (for PC400) depending on how the host presentation space is configured.

For the **Convert RowCol** function (**R** as the second character in the calling data string), a value of 0 indicates an error in the input value for row (calling length parameter).

Return Code:

The **Convert Position or RowCol** function is the exception to the rule that the fourth return parameter always contains a return code. For this function, the value returned in the fourth parameter is called a status code. This status code can contain data or a return code. Your application must provide for processing of this status code to prevent unpredictable results or an error.

- If the value of the fourth parameter is 0, 9998, or 9999, it is a return code.
- For the **Convert Position** function (**P** as the second character of the calling data string), a value in the range of 1–132 is the number of the column that contains the PS position passed in the

calling PS Position parameter. The upper limit can be smaller than 132 depending on how the host presentation space is configured.

- For the **Convert RowCol** function (R as the second character of the calling data string), a value in the range of 1–3564 represents the host presentation space position that corresponds to the row and column values passed in the calling length and PS position parameters, respectively. The upper limit can be smaller than 3564 depending on how the host presentation space is configured.

The following status codes are defined:

Status Code	Explanation
0	This is an incorrect PS position or column.
>0	This is the PS position or column.
9998	An incorrect host presentation space ID was specified or a system error occurred.
9999	Character 2 in the data string is not P or R.

Notes on Using This Function

- To configure your presentation space, refer to *Administrator's Guide and Reference*
- To find out how many rows and columns are in your presentation space, examine the returned data string parameter for the **Query Session Status** (22) function. See [Query Session Status \(22\) on page 107](#).

Copy Field to String (34)

3270	5250	VT
Yes	Yes	Yes

The **Copy Field to String** function transfers characters from a field in the host-connected presentation space into a string.

The **Copy Field to String** function translates the characters in the host source presentation space into American National Standard Code for Information Interchange (ASCII). Attribute bytes and other characters not represented in ASCII normally are translated into blanks.

Prerequisite Calls

Connect Presentation Space (1)

Call Parameters

	Standard Interface	Enhanced Interface
Function Number	Must be 34	

	Standard Interface	Enhanced Interface
Data String	Preallocated target data string. When the Set Session Parameters (9) function with Extended Attribute Bytes (EAB) option is issued, the length of the data string must be at least twice the length of the field.	
Length	Number of bytes to copy (the length of the data string).	
PS Position	Identifies the target field. This can be the PS position of any byte within the target field. Copy always starts at the beginning of the field.	

Return Parameters

This function returns a data string, length, and a return code.

Data String:

A string containing data from the identified field in the host presentation space. The first byte in the returned data string is the beginning byte of the identified field in the host presentation space. The number of bytes in the returned data string is determined by the smaller of:

- Number of bytes specified in the calling length parameter
- Number of bytes in the identified field in the host presentation space

Length:

The length of the data returned.

Return Code:

The following codes are defined:

Return Code	Explanation
0	The Copy Field to String function was successful.
1	Your program is not connected to a host session.
2	An error was made in specifying parameters.
6	The data to be copied and the target field are not the same size. The data is truncated if the string length is smaller than the field copied.
7	The host presentation space position is not valid.
9	A system error was encountered.
24	Unformatted host presentation space.

Notes on Using This Function

1. The field position and length information can be found by using the **Find Field Position** (31) and **Find Field Length** (32) functions. The **Copy Field to String** function can be used with either protected or unprotected fields, but only in a *field-formatted* host presentation space.
2. The copy is ended when one of the following conditions is encountered:

- When the end of the field is reached
 - When the length of the target string is exceeded
3. An EAB can be returned when the **Set Session Parameters** (9) function EAB option is used. EAB is related to each character in the presentation space and is returned preceding each character.
 4. The **Copy Field to String** function is affected by the `ATTRB/NOATTRB/NULLATTRB`, the `EAB/NOEAB`, the `XLATE/NOXLATE`, the `DISPLAY/NODISPLAY`, the `DISPATTR/NODISPATTR` session options. Refer to items [5 on page 142](#); [13 on page 145](#) and [14 on page 146](#); [17 on page 146](#); and `#unique_169_Connect_42_opt18` and `#unique_169_Connect_42_opt19` for more information.

As previously stated, the return of attributes by the various **Copy** (5, 8, and 34) functions is affected by the **Set Session Parameters** (9) function. The involved set session parameters have the following effect:

Set Session Parameter

Effect on the COPY Function

NOEAB and NOEAD

Attributes are not returned. Only text is copied from the presentation space to the user buffer.

EAB and NOXLATE

Attributes are returned as defined in the following tables.

EAB and XLATE

The colors used for the presentation space display are returned. Colors can be remapped; so the attribute colors are not the ones returned by the **COPY** functions when XLATE and EAB are on at the same time.

The returned character attributes are defined in the following tables. The attribute bit positions are in IBM® format with bit 0 the left most bit in the byte.

3270 character attributes are returned from the host to the emulator. The following table applies when EAB and NOXLATE are set.

Bit Position	Meaning
0–1	Character highlighting 00 = Normal 01 = Blink 10 = Reverse video 11 = Underline
2–4	Character color (Color remap can override this color definition.) 000 = Default 001 = Blue 010 = Red 011 = Pink 100 = Green

Bit Position	Meaning
	101 = Turquoise 110 = Yellow 111 = White
5–6	Character attributes 00 = Default value
7	Reserved

5250 character attributes are returned from the host to the emulator. The following table applies when EAB and NOXLATE are set.

Bit Position	Meaning
0	Reverse image 0 = Normal image 1 = Reverse image
1	Underline 0 = No underline 1 = Underline
2	Blink 0 = Not blink 1 = Blink
3	Separator of columns 0 = No separator 1 = Separator
4–7	Reserved

The following table shows Z and I Emulator for Windows character color attributes. The following table applies when EAB and XLATE are set.

Bit Position	Meaning
0–3	Background character colors 0000 = Black 0001 = Blue 0010 = Green 0011 = Cyan 0100 = Red 0101 = Magenta 0110 = Brown (3270), Yellow (5250) 0111 = White
4–7	Foreground character colors

Bit Position	Meaning
	0000 = Black 0001 = Blue 0010 = Green 0011 = Cyan 0100 = Red 0101 = Magenta 0110 = Brown (3270), Yellow (5250) 0111 = White 1000 = Gray 1001 = Light blue 1010 = Light green 1011 = Light cyan 1100 = Light red 1101 = Light magenta 1110 = Yellow 1111 = White (high intensity)

For a PS/2® monochrome display, the characters in the application (workstation) session appear as various shades of gray. This is required to give users their remapped colors in the EHLLAPI application session so they can get what they see in their host application presentation spaces.

- To use this function, preallocate memory to receive the returned data string parameter. The statements required to preallocate this memory vary depending on the language in which your application is written. Refer to [Memory Allocation on page 10](#) for more information.



Note: 5250 emulation supports a presentation space of 24 rows by 80 columns. In some instances, Communication Manager 5250 emulation displays a 25th row. This occurs when either an error message from the host is displayed or when the operator selects the SysReq key. Z and I Emulator for Windows displays 25th row information on the status bar. By **EXTEND_PS** option, an EHLLAPI application can use the same interface with Communication Manager EHLLAPI and valid presentation space is extended when this condition occurs.

Copy OIA (13)

3270	5250	VT
Yes	Yes	Yes

The **Copy OIA** function returns the current operator information area (OIA) data from the host-connected presentation space.

The OIA is located under the bottom dividing line of the screen and is used to display session status information about the connection between the workstation and the host.

Prerequisite Calls

Connect Presentation Space (1)

Call Parameters

	Standard Interface	Enhanced Interface
Function Number	Must be 13	
Data String	Preallocated target data string	
Length	103	104
PS Position	NA	

Return Parameters

This function returns a data string and a return code.

Data String:

A 103-byte string for 16-bit and 104-byte string for 32-bit. See [Format of the Returned OIA Data String on page 51](#) for more information.

Return Code:

The following codes are defined:

Return Code	Explanation
0	OIA data is returned. The target presentation space is unlocked.
1	Your program is not connected to a host session.
2	An error was made in specifying string length. OIA data was not returned.
4	OIA data is returned. The target presentation space is busy.
5	OIA data is returned. The target presentation space is locked. (Input inhibited)
9	An internal system error was encountered. OIA data was not returned.

Notes on Using This Function

1. The OIA Group consists of the bits that show the status of the connected sessions. The group is categorized by the represented host function. (For example, Group 8 consists of the bits that show all conditions of the input inhibit in the session.) The states of each group are ordered so that the high-order bits represent the

indicators of higher priority. That is, bit 7 has priority over bit 0. Therefore, if more than one state is active within a group, the state with the highest priority is the active state within that group.

2. To use this function, preallocate memory to receive the returned data string parameter. The statements required to preallocate this memory vary depending on the language in which your application is written. Refer to [Memory Allocation on page 10](#) for more information.

Format of the Returned OIA Data String

The OIA data string contains the following information:

Byte		Definition
Standard	Enhanced	
1	1	The OIA format byte. The value is 1 (PC/3270), 9 (PC400), or 5 (VT).
2–81	2–81	The OIA image in the host code points.
82–103	82–103	OIA group indicator meanings.
	104	Reserved.

PC/3270 OIA Group Indicator Meanings and Its Image

The OIA image group consists of an 80-byte ASCII character string with no attribute bytes that contains the OIA image in host code points. [Figure 2: Host Presentation Space Characters on page 52](#) shows the hexadecimal codes found in the host presentation space, and the characters they represent. The returned data can be translated into OIA graphics characters. Refer to *Quick Beginnings* for information on the OIA indicators.

To translate the returned data into OIA graphics characters, proceed as follows:

1. Print the data returned in bytes 2 through 81 to the screen or to a printer.
2. Using the code page chart applicable to the device on which the output appears, find the hexadecimal value corresponding to each character.
3. Using [Figure 2: Host Presentation Space Characters on page 52](#), find the OIA graphics character corresponding to each hexadecimal value found in step 2.



Note: Group 8 (byte 0) machine, communications, and program check images are followed by a three-digit number related to the type of check.

The short session ID followed by X'20' is in column 7.

All group images are represented by Main Frame Interactive (MFI) hex code points.



Note: The OIA image data string position minus 1 position equals the OIA column.

Figure 2. Host Presentation Space Characters

	0x	1x	2x	3x	4x	5x	6x	7x	8x	9x	Ax	Bx	Cx	Dx	Ex	Fx
x0	NUL	SP	0	&	à	ã	À	Ã	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	z	a	→	↔
x3	NL	"	3	,	ò	ó	Ò	Ó	d	t	D	T	—	°	↑	↗
x4	STP	/	4	:	ù	ü	Ù	Ü	e	u	E	U	⋮	°	↑	4
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	[\$	β	^	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	À	⌋	⌋	□	☰
xD	(¥	@	`	Ù	ó	U	Ó	n	ç	N	Ç	B	⌋	↔	□
xE	}	Pts	%	'	Ü	Ú	Y	Ú	o	;	O	;	·	+	□	i
xF	{	☀	—	,	Ç	ñ	C	Ñ	p	*	P	*	■	×	■	Not Supported

- Group 1 (Offset 82): Online and Screen Ownership

Bit	Meaning
0–1	Reserved
2	SSCP-LU session owns screen
3	LU-LU session owns screen
4	Online and not owned
5	Subsystem ready

Bit	Meaning
6–7	Reserved

- Group 2 (Offset 83): Character Selection

Bit	Meaning
0	Reserved
1	APL
3	Alphanumeric
4–5	Reserved

- Group 3 (Offset 84): Shift State

Bit	Meaning
0	Upper shift
1	Numeric
2	CAPS
3–7	Reserved

- Group 4 (Offset 85): PSS Group 1

Bit	Meaning
0–7	Reserved

- Group 5 (Offset 86): Highlight Group 1

Bit	Meaning
0	Operator selectable
1	Field inherit
2–7	Reserved

- Group 6 (Offset 87): Color Group 1

Bit	Meaning
0	Operator selectable
1	Field inherit
2–7	Reserved

- Group 7 (Offset 88): Insert

Bit	Meaning
0	Insert mode
1–7	Reserved

- Group 8 (Offset 89–93): Input Inhibited (5 bytes)

◦ Byte 1 (Offset 89)

Bit	Meaning
0	Non-resettable machine check
1	Reserved
2	Machine check
3	Communications check
4	Program check
5-7	Reserved

◦ Byte 2 (Offset 90)

Bit	Meaning
0	Device busy
1	Terminal wait
2	Minus symbol
3	Minus function
4	Too much entered
5-7	Reserved

◦ Byte 3 (Offset 91)

Bit	Meaning
0-2	Reserved
3	Incorrect dead key combination, limited key.
4	Wrong place
5-7	Reserved

◦ Byte 4 (Offset 92)

Bit	Meaning
0-1	Reserved
2	System wait
3-7	Reserved

◦ Byte 5 (Offset 93)

Bit	Meaning
0-7	Reserved

• Group 9 (Offset 94): PSS Group 2

Bit	Meaning
0-7	Reserved

- Group 10 (Offset 95): Highlight Group 2

Bit	Meaning
0–7	Reserved

- Group 11 (Offset 96): Color Group 2

Bit	Meaning
0–7	Reserved

- Group 12 (Offset 97): Communication Error Reminder

Bit	Meaning
0-6	Communications error
1–7	Reserved

- Group 13 (Offset 98): Printer State

Bit	Meaning
0–7	Reserved

- Group 14 (Offset 99): Graphics

Bit	Meaning
0–7	Reserved

- Group 15 (Offset 100): Reserved
- Group 16 (Offset 101): Automatic Key Play/Record State

Bit	Meaning
0–7	Reserved

- Group 17 (Offset 102): Automatic Key Quit/Stop State

Bit	Meaning
0–7	Reserved

- Group 18 (Offset 103): Expanded State

Bit	Meaning
0–7	Reserved

PC400 OIA Group Indicator Meanings and Its Image

Details of the OIA group are listed in the following tables.

- Group 1 (Offset 82): Online and Screen Ownership

Bit	Meaning	Beginning Position of Data String
0–2	Reserved	
3	System available	1
4	Reserved	
5	Subsystem ready	
6–7	Reserved	

- Group 2 (Offset 83): Character Selection

Bit	Meaning	Beginning Position of Data String
0–1	Reserved	
3	Alphanumeric	
4–5	Reserved	

- Group 3 (Offset 84): Shift State

Bit	Meaning	Beginning Position of Data String
0	Reserved	
1	Keyboard shift	39
2	CAPS	
3–6	Reserved	

- Group 4 (Offset 85): PSS Group 1

Bit	Meaning	Beginning Position of Data String
0–7	Reserved	

- Group 5 (Offset 86): Highlight Group 1

Bit	Meaning	Beginning Position of Data String
0–7	Reserved	

- Group 6 (Offset 87): Color Group 1

Bit	Meaning	Beginning Position of Data String
0–7	Reserved	

- Group 7 (Offset 88): Insert

Bit	Meaning	Beginning Position of Data String
0	Insert mode	68
1–7	Reserved	

- Group 8 (Offset 89–93): Input Inhibited (5 bytes)

◦ Byte 1 (Offset 89)

Bit	Meaning	Beginning Position of Data String
0–7	Reserved	

◦ Byte 2 (Offset 90)

Bit	Meaning	Beginning Position of Data String
0–7	Reserved	

◦ Byte 3 (Offset 91)

Bit	Meaning	Beginning Position of Data String
0–4	Reserved	
5	Operator input error	64
6–7	Reserved	

◦ Byte 4 (Offset 92)

Bit	Meaning	Beginning Position of Data String
0–1	Reserved	
2	System wait	64
3–7	Reserved	

◦ Byte 5 (Offset 93)

Bit	Meaning	Beginning Position of Data String
0–7	Reserved	

• Group 9 (Offset 94): PSS Group 2

Bit	Meaning	Beginning Position of Data String
0–7	Reserved	

• Group 10 (Offset 95): Highlight Group 2

Bit	Meaning	Beginning Position of Data String
0–7	Reserved	

• Group 11 (Offset 96): Color Group 2

Bit	Meaning	Beginning Position of Data String
0–7	Reserved	

• Group 12 (Offset 97): Communication Error Reminder

Bit	Meaning	Beginning Position of Data String
0	Communications Error	
1–5	Reserved	
7	Message wait	3

- Group 13 (Offset 98): Printer State

Bit	Meaning	Beginning Position of Data String
0–7	Reserved	

- Group 14 (Offset 99): Graphics

Bit	Meaning	Beginning Position of Data String
0–7	Reserved	

- Group 15 (Offset 100): Reserved
- Group 16 (Offset 101): Automatic Key Play/Record State

Bit	Meaning	Beginning Position of Data String
0–7	Reserved	

- Group 17 (Offset 102): Automatic Key Quit/Stop State

Bit	Meaning	Beginning Position of Data String
0–7	Reserved	

- Group 18 (Offset 103): Expanded State

Bit	Meaning	Beginning Position of Data String
0–7	Reserved	

VT Host OIA Group Indicator Meanings and Its Image

Details of the VT Host OIA group are listed in the following tables.

- Group 1 (Offset 82): Online and Screen Ownership

Bit	Meaning
5	Subsystem ready

- Group 2 (Offset 83): Character Selection

Bit	Meaning
0	Upper shift
2	CAPS

- Group 7 (Offset 88): Insert

Bit	Meaning
0	Insert mode

Some columns on the OIA line display different messages for VT than those messages displayed for 3270/5250. See the following table for specific details.

Column	Symbol
1-7	VT220 7
	VT220 8
	VT100
	VT52
	VTANSI
9 - 12	LOCK
61 - 64	HOLD

Copy Presentation Space (5)

3270	5250	VT
Yes	Yes	Yes

The **Copy Presentation Space** function copies the contents of the host-connected presentation space into a data string that you define in your EHLLAPI application program.

The **Copy Presentation Space** function translates the characters in the host source presentation space into ASCII. Attribute bytes and other characters not represented in ASCII normally are translated into blanks. If you do not want the attribute bytes translated into blanks, you can override this translation with the ATTRB option under the **Set Session Parameters (9)** function.

Prerequisite Calls

Connect Presentation Space (1)

Call Parameters

	Standard Interface	Enhanced Interface
Function Number	Must be 5	
Data String	Preallocated target string the size of your host presentation space. This can vary depending on how your host presentation space is configured. When the Set Session Parameters (9) function with the EAB option is issued, the length of the data string must be at least twice the length of the presentation space.	
Length	NA (the length of the host presentation space is implied).	

	Standard Interface	Enhanced Interface
PS Position	NA.	

Return Parameters

This function returns a data string, length, and a return code.

Data String:

Contents of the connected host presentation space.

Length:

Length of the data copied.

Return Code:

The following codes are defined:

Return Code	Explanation
0	The host presentation space contents were copied to the application program. The target presentation space was active, and the keyboard was unlocked.
1	Your program is not connected to a host session.
4	The host presentation space contents were copied. The connected host presentation space was waiting for host response.
5	The host presentation space was copied. The keyboard was locked.
9	A system error was encountered.

Notes on Using This Function

1. An EAB can be returned when the **Set Session Parameters** (9) function EAB option is used. EAB is related to each character in the presentation space and is returned preceding each character.
2. The **Copy Presentation Space** function is affected by the following session options:
 - ATTRB/NOATTRB/NULLATTRB
 - EAB/NOEAB
 - XLATE/NOXLATE
 - BLANK/NOBLANK
 - DISPLAY/NODISPLAY
 - EXTEND_PS/NOEXTEND_PS

Refer to items [5 on page 142](#); [13 on page 145](#), [14 on page 146](#), [15 on page 146](#) and [17 on page 146](#); and #unique_169_Connect_42_opt18 and #unique_169_Connect_42_opt19 for more information.

If the target data string provided is not long enough to hold the requested data, unpredictable results can occur.

As previously stated, the return of attributes by the various **Copy** (5, 8, and 34) functions is affected by the **Set Session Parameters** (9) function. The involved set session parameters have the following effect:

Set Session Parameter**Effect on the COPY Function****NOEAB and NOEAD**

Attributes are not returned. Only text is copied from the presentation space to the user buffer.

EAB and NOXLATE

Attributes are returned as defined in the following tables.

EAB and XLATE

The colors used for the presentation space display are returned. Colors can be remapped; so the attribute colors are not the ones returned by the **Copy** functions when XLATE and EAB are on at the same time.

NOSO/SPACESO/SO

When NOSO is specified, it works as SPACESO. The size of the presentation space is not changed.

The returned character attributes are defined in the following tables. The attribute bit positions are in IBM® format with bit 0 the left most bit in the byte.

3270 character attributes are returned from the host to the emulator. The following table applies when EAB and NOXLATE are set.

Bit Position	Meaning
0–1	Character highlighting 00 = Normal 01 = Blink 10 = Reverse video 11 = Underline
2–4	Character color (Color remap can override this color definition.) 000 = Default 001 = Blue 010 = Red 011 = Pink 100 = Green 101 = Turquoise 110 = Yellow 111 = White
5–6	Character attribute

Bit Position	Meaning
	00 = Default value
7	Reserved

5250 character attributes are returned from the host to the emulator. The following table applies when EAB and NOXLATE are set.

Bit Position	Meaning
0	Reverse image 0 = Normal image 1 = Reverse image
1	Underline 0 = No underline 1 = Underline
2	Blink 0 = Not blink 1 = Blink
3	Separator of columns 0 = No separator 1 = Separator
4–7	Reserved

The following table shows Z and I Emulator for Windows character color attributes. The following table applies when EAB and XLATE are set.

Bit Position	Meaning
0–3	Background character colors 0000 = Black 0001 = Blue 0010 = Green 0011 = Cyan 0100 = Red 0101 = Magenta

Bit Position	Meaning
	0110 = Brown (3270), Yellow (5250) 0111 = White
4–7	Foreground character colors 0000 = Black 0001 = Blue 0010 = Green 0011 = Cyan 0100 = Red 0101 = Magenta 0110 = Brown (3270), Yellow (5250) 0111 = White 1000 = Gray 1001 = Light blue 1010 = Light green 1011 = Light cyan 1100 = Light red 1101 = Light magenta 1110 = Yellow 1111 = White (high intensity)

For a PS/2® monochrome display, the characters in the application (workstation) session appear as various shades of gray. This is required to give users their remapped colors in the EHLLAPI application session so they can get what they see in their host application presentation spaces.

If you want to copy only a portion of the host presentation space, use the **Copy Presentation Space to String (8)** function.

To use this function, preallocate memory to receive the returned data string parameter. The statements required to preallocate this memory vary depending on the language in which your application is written. Refer to [Memory Allocation on page 10](#) for more information.



Note: 5250 emulation supports a presentation space of 24 rows by 80 columns. In some instances, Communication Manager 5250 emulation displays a 25th row. This occurs when either an error message from the host is displayed or when the operator selects the SysReq key. Z and I Emulator for Windows displays 25th row information on row 24, or on the status bar. For information to be displayed on the status bar, the status bar must be configured. Refer to *Quick Beginnings* for information on configuring the status



bar. By the **EXTEND_PS** option, an EHLLAPI application can use the same interface with Communication Manager EHLLAPI and valid presentation space is extended when this condition occurs.

Copy Presentation Space to String (8)

3270	5250	VT
Yes	Yes	Yes

The **Copy Presentation Space to String** function is used to copy all or part of the host-connected presentation space into a data string that you define in your EHLLAPI application program.

The input PS position is the offset into the host presentation space. This offset is based on a layout in which the upper-left corner (row 1/column 1) is location 1 and the bottom-right corner is 3564, which is the maximum screen size for the host presentation space. The value of `PS Position + (Length - 1)` cannot exceed the configured size of your host presentation space.

The **Copy Presentation Space to String** function translates the characters in the host source presentation space into ASCII. Attribute bytes and other characters not represented in ASCII normally are translated into blanks. If you do not want the attribute bytes translated into blanks, you can override this translation with the ATTRB option under the **Set Session Parameters (9)** function.

Prerequisite Calls

Connect Presentation Space (1).

Call Parameters

	Standard Interface	Enhanced Interface
Function Number	Must be 8	
Data String	Preallocated target string the size of your host presentation space. When the Set Session Parameters (9) function with the EAB option is issued, the length of the data string must be at least twice the length of the presentation space.	
Length	Length of the target data string.	
PS Position	Position within the host presentation space of the first byte in your target data string.	

Return Parameters

This function returns a data string and a return code.

Data String:

Contents of the host presentation space.

Return Code:

The following codes are defined:

Return Code	Explanation
0	The host presentation space contents were copied to the application program. The target presentation space was active, and the keyboard was unlocked.
1	Your program is not connected to a host session.
2	An error was made in specifying string length, or the sum of (Length - 1) + PS position is greater than the size of the connected host presentation space.
4	The host presentation space contents were copied. The host presentation space was waiting for host response.
5	The host presentation space was copied. The keyboard was locked.
7	The host presentation space position is not valid.
9	A system error was encountered.

Notes on Using This Function

1. An EAB can be returned when the **Set Session Parameters** (9) function EAB option is used. EAB is related to each character in the presentation space and is returned following each character.
2. The **Copy Presentation Space to String** function is affected by the following options:

- ATTRB/NOATTRB/NULLATTRB
- EAB/NOEAB
- XLATE/NOXLATE
- BLANK/NOBLANK
- DISPLAY/NODISPLAY
- EXTEND_PS/NOEXTEND_PS

Refer to items [5 on page 142](#); [13 on page 145](#) and [14 on page 146](#); [15 on page 146](#); [17 on page 146](#); and #unique_169_Connect_42_opt18 and #unique_169_Connect_42_opt19

If the target data string provided is not large enough to hold the requested number of bytes, the copy ends successfully (RC=0, 4, or 5) when the end of the target data string is reached.

As previously stated, the return of attributes by the various **Copy** (5, 8, and 34) functions is affected by the **Set Session Parameters** (9) function. The involved set session parameters have the following effect:

Set Session Parameter

Effect on the Copy Function

NOEAB and NOEAD

Attributes are not returned. Only text is copied from the presentation space to the user buffer.

EAB and NOXLATE

Attributes are returned as defined in the following tables.

EAB and XLATE

The colors used for the presentation space display are returned. Colors can be remapped, so the attribute colors are not the ones returned by the **Copy** functions when XLATE and EAB are on at the same time.

The returned character attributes are defined in the following tables. The attribute bit positions are in IBM format with bit 0 the left most bit in the byte.

- 3270 character attributes are returned from the host to the emulator. The following table applies when EAB and NOXLATE are set.

Bit Position	Meaning
0–1	Character highlighting 00 = Normal 01 = Blink 10 = Reverse video 11 = Underline
2–4	Character color (Color remap can override this color definition.) 000 = Default 001 = Blue 010 = Red 011 = Pink 100 = Green 101 = Turquoise 110 = Yellow 111 = White
5–7	Reserved

- 5250 character attributes are returned from the host to the emulator. The following table applies when EAB and NOXLATE are set.

Bit Position	Meaning
0	Reverse image 0 = Normal image 1 = Reverse image
1	Underline 0 = No underline 1 = Underline
2	Blink 0 = Not blink 1 = Blink

Bit Position	Meaning
3	Separator of columns 0 = No separator 1 = Separator
4–7	Reserved

- VT character attributes are returned from the host to the emulator. The following table applies when EAB and NOXLATE are set.

Bit Position	Meaning
0-3	Reserved
4	Bold 1 = On 0 = Off
5	Underscore 1 = On 1 = Off
6	Blink 1 = On 0 = Off
7	Reverse 0 = On 1 = Off

- The following table shows Z and I Emulator for Windows character color attributes. The following table applies when EAB and XLATE are set.

Bit Position	Meaning
0–3	Background character colors 0000 = Black 0001 = Blue 0010 = Green 0011 = Cyan 0100 = Red 0101 = Magenta 0110 = Brown (3270), Yellow (5250) 0111 = White
4–7	Foreground character colors 0000 = Black 0001 = Blue 0010 = Green

Bit Position	Meaning
	0011 = Cyan
	0100 = Red
	0101 = Magenta
	0110 = Brown (3270), Yellow (5250)
	0111 = White
	1000 = Gray
	1001 = Light blue
	1010 = Light green
	1011 = Light cyan
	1100 = Light red
	1101 = Light magenta
	1110 = Yellow
	1111 = White (high intensity)

For a PS/2 monochrome display, the characters in the application (workstation) session appear as various shades of gray. This is required to give users their remapped colors in the EHLLAPI application session so they can get what they see in their host application presentation spaces.

- To use this function, preallocate memory to receive the returned data string parameter. The statements required to preallocate this memory vary depending on the language in which your application is written. Refer to [Memory Allocation on page 10](#) for more information.



Note: 5250 emulation supports a presentation space of 24 rows by 80 columns. In some instances, Communication Manager 5250 emulation displays a 25th row. This occurs when either an error message from the host is displayed or when the operator selects the SysReq key. Z and I Emulator for Windows displays 25th row information on row 24, or on the status bar. For information to be displayed on the status bar, the status bar must be configured. Refer to *Quick Beginnings* for information on configuring the status bar. By the **EXTEND_PS** option, an EHLLAPI application can use the same interface with Communication Manager EHLLAPI and valid presentation space is extended when this condition occurs.

Copy String to Field (33)

3270	5250	VT
Yes	Yes	Yes

The **Copy String to Field** function transfers a string of characters into a specified field in the host-connected presentation space. This function can be used only in a *field-formatted* host presentation space.

Prerequisite Calls

Connect Presentation Space (1)

Call Parameters

	Standard Interface	Enhanced Interface
Function Number	Must be 33	
Data String	String containing the data to be transferred to a target field in the host presentation space.	
Length	Length, in number of bytes, of the source data string. Overridden if in EOT mode.	
PS Position	Identifies the target field. This can be the PS position of any byte within the target field. Copy always starts at the beginning of the field.	

Return Parameters

Return Code	Explanation
0	The Copy String to Field function was successful.
1	Your program is not connected to a host session.
2	Parameter error or zero length for copy.
5	The target field was protected or inhibited, or incorrect data was sent to the target field (such as a field attribute).
6	Copy was completed, but data is truncated.
7	The host presentation space position is not valid.
9	A system error was encountered.
24	Unformatted host presentation space.

Notes on Using This Function

1. The **Copy String to Field** function is affected by the following options:

- STRLEN/STREOT
- EOT
- EAB/NOEAB
- XLATE/NOXLATE
- PUTEAB/NOPUTEAB

Refer to items [1 on page 141](#) and [2 on page 141](#); [13 on page 145](#) and [14 on page 146](#); [18 on page 146](#); and #unique_169_Connect_42_opt18 and #unique_169_Connect_42_opt19 for more information.

2. The string to be transferred is specified with the calling data string parameter. The string ends when one of these three conditions is encountered:

- When an end-of-text (EOT) delimiter is encountered in the string if EOT mode was selected using the **Set Session Parameters** (9) function. (See [Set Session Parameters \(9\) on page 139](#)).
- When the number specified in the length is reached if not in EOT mode.
- When an end-of-field is encountered in the field.



Note: If the field at the end of the host presentation space wraps, wrapping occurs when the end of the presentation space is reached.

3. The keyboard mnemonics (see **Send Key** (3) function) cannot be sent using the **Copy String to Field** function.
4. The first byte of the data to be transferred is always placed at the beginning of the field that contains the specified PS position.



Note: 5250 emulation supports a presentation space of 24 rows by 80 columns. In some instances, Communication Manager 5250 emulation displays a 25th row. This occurs when either an error message from the host is displayed or when the operator selects the SysReq key. Z and I Emulator for Windows displays 25th row information on row 24, or on the status bar. For information to be displayed on the status bar, the status bar must be configured. Refer to *Quick Beginnings* for information on configuring the status bar. By the **EXTEND_PS** option, an EHLLAPI application can use the same interface with Communication Manager EHLLAPI and valid presentation space is extended when this condition occurs.

Copy String to Presentation Space (15)

3270	5250	VT
Yes	Yes	Yes

The **Copy String to Presentation Space** function copies an ASCII data string directly into the host presentation space at the location specified by the PS position calling parameter.

Prerequisite Calls

Connect Presentation Space (1)

Call Parameters

	Standard Interface	Enhanced Interface
Function Number	Must be 15.	
Data String	String of ASCII data to be copied into the host presentation space.	
Length	Length, in number of bytes, of the source data string. Overridden if in EOT mode.	
PS Position	Position in the host presentation space to begin the copy, a value between 1 and the configured size of your host presentation space.	

Return Parameters

Return Code	Explanation
0	The Copy String to Presentation Space function was successful.
1	Your program is not connected to a host session.
2	Parameter error or zero length for copy.
5	The target presentation space is protected or inhibited, or incorrect data was sent to the target presentation space (such as a field attribute byte).
6	The copy was completed, but the data was truncated.
7	The host presentation space position is not valid.
9	A system error was encountered.

Notes on Using This Function

1. The **Copy String to Presentation Space** function is affected by the following options:

- STRLEN/STREOT
- EOT
- EAB/NOEAB
- XLATE/NOXLATE
- PUTEAB/NOPUTEAB
- EXTEND_PS/NOEXTEND_PS

Refer to items [1 on page 141](#) and [2 on page 141](#); [13 on page 145](#) and [14 on page 146](#); [18 on page 146](#); and #unique_169_Connect_42_opt18 and #unique_169_Connect_42_opt19 for more information.

2. The keyboard mnemonics (see **Send Key** (3) function) cannot be sent using the **Copy String to Presentation Space** function.
3. The string ends when an end-of-text (EOT) delimiter is encountered in the string if EOT mode was selected using the **Set Session Parameters** (9) function. (See [Set Session Parameters \(9\) on page 139](#)).
4. Although the **Send Key** (3) function accomplishes the same purpose, this function responds with the prompt and enters a command more quickly. Because the **Send Key** (3) function emulates the terminal operator typing the data from the keyboard, its process speed is slow for an application operating with a lot of data. This function provides a faster input path to the host.
5. The original data (the copied string) cannot exceed the size of the presentation space.

6. This function call may cause a cursor movement to an unexpected position with some host applications. A SendKey function may be a better choice for filling a field than this function.



Note: This only occurs with VT sessions or connections to an ASCII host.

Copy Presentation Space to Clipboard (35)

3270	5250	VT
Yes	Yes	Yes

The **Copy Presentation Space to Clipboard** function is used to copy all or part of the host-connected presentation space into clipboard. The input PS position is the offset into the host presentation space. This offset is based on a layout in which the upper-left corner (row 1/column 1) is location 1 and the bottom-right corner is 3564, which is the maximum screen size for the host presentation space. The value of PS Position + (Length – 1) cannot exceed the configured size of your host presentation space.

The **Copy Presentation Space to Clipboard** translates the characters in the host source presentation space into ASCII. Attribute bytes and other characters not represented in ASCII normally are translated into blanks. If you do not want the attribute bytes translated into blanks, you can override this translation with the ATTRB option under the **Set Session Parameters (9)** function.

Prerequisite Calls

Connect Presentation Space (1)

Call Parameters

	Standard Interface	Enhanced Interface
Function Number	Must be 35.	
Data String	Preallocated target string the size of your host presentation space. When the Set Session Parameters (9) function with the EAB option is issued, the length of the data string must be at least twice the length of the presentation space.	
Length	Length of the target data string	
PS Position	Position within the host presentation space of the first byte in your target data string.	

Return Parameters

Return Code	Explanation
0	The host presentation space contents were copied to the clipboard. The target presentation space was active, and the keyboard was unlocked.
1	Your program is not connected to a host session.

Return Code	Explanation
2	An error was made in specifying string length, or the sum of (Length - 1) + PS position is greater than the size of the connected host presentation space.
4	The host presentation space contents were copied to clipboard. The host presentation space was waiting for host response.
5	The host presentation space was copied to clipboard. The keyboard was locked.
7	The host presentation space position is not valid.
9	A system error was encountered.

Notes on Using This Function

1. An EAB can be returned when the **Set Session Parameters** (9) function EAB option is used. EAB is related to each character in the presentation space and is returned following each character.
2. The **Copy Presentation Space to Clipboard** function is affected by the following options:
 - ATTRB/NOATTRB/NULLATTRB
 - EAB/NOEAB
 - XLATE/NOXLATE
 - BLANK/NOBLANK
 - DISPLAY/NODISPLAY
 - PUTEAB/NOPUTEAB
 - EXTEND_PS/NOEXTEND_PS
3. The data string buffer is used in processing the data retrieved from presentation space and copy to clipboard. If the data string buffer provided is not large enough to hold the requested number of bytes, the copy ends successfully (RC=0, 4, or 5) when the end of the data string buffer is reached. As previously stated, the return of attributes by the various **Copy** (5, 8, and 34) functions is affected by the **Set Session Parameters** (9) function. The involved set session parameters have the following effect:

Set Session Parameter

Effect on the Copy Function

NOEAB and NOEAD

Attributes are not returned. Only text is copied from the presentation space to the clipboard.

EAB and NOXLATE

Attributes are returned as defined in the following tables.

EAB and XLATE

The colors used for the presentation space display are returned. Colors can be remapped, so the attribute colors are not the ones returned by the **Copy** functions when XLATE and EAB are on at the same time.

Paste Clipboard to Presentation Space (36)

3270	5250	VT
Yes	Yes	Yes

The **Paste Clipboard to Presentation Space** function pastes an ASCII data string directly into the host presentation space at the location specified by the PS position calling parameter.

Prerequisite Calls

Call Parameters

	Standard Interface	Enhanced Interface
Function Number	Must be 36.	
Data String	String buffer that holds the data from clipboard to paste into the host presentation space.	
Length	Length, in number of bytes to be pasted	
PS Position	Position in the host presentation space to begin the copy, a value between 1 and the configured size of your host presentation space.	


Return Parameters

Return Code	Explanation
0	The Paste Clipboard to Presentation Space function was successful.
1	Your program is not connected to a host session.
2	Parameter error or zero length for copy
5	The target presentation space is protected or inhibited, or incorrect data was sent to the target presentation space (such as a field attribute byte).
6	The copy was completed, but the data was truncated.
7	The copy was completed, but the data was truncated.
9	A system error was encountered.

Notes on Using This Function

- The **Paste Clipboard to Presentation Space** function is affected by the following options:
 - STRLEN/STREOT
 - EAB/NOEAB
 - EOT
 - XLATE/NOXLATE

- PUTEAB/NOPUTEAB
 - EXTEND_PS/NOEXTEND_PS
2. The string ends when an end-of-text (EOT) delimiter is encountered in the string if EOT mode was selected using the **Set Session Parameters (9)** function. (See “Set Session Parameters (9)” on page 147).
 3. The original data (the copied string) cannot exceed the size of the presentation space.

String	Meanings	Single-byte character field	
X'000C'	(NULL)(FF) X'00'X'0C'	(SB NULL)(SB FF) X'00'X'0C'	
X'0E000C0F'	(SO)(DB FF)(SI) X'0E'X'000C'X'0F'	–S error	
 Note: SB means single-byte characters.			

Note: 5250 emulation supports a presentation space of 24 rows by 80 columns. In some instances, Communication Manager 5250 emulation displays a 25th row. This occurs when either an error message from the host is displayed or when the operator selects the SysReq key. Z and I Emulator for Windows always displays the same information on the 24th row. By the **EXTEND_PS** option, an EHLLAPI application can use the same interface with Communication Manager EHLLAPI and valid presentation space is extended when this condition occurs.

Disconnect from Structured Fields (121)

3270	5250	VT
Yes	No	No

The **Disconnect from Structured Fields** function drops the connection between the emulation program and the EHLLAPI application. The EHLLAPI application must disconnect from the emulation program before exiting from the system. The EHLLAPI application should issue this function request if a previous **Connect for Structured Fields** was issued.

The **Reset System (21)** function will also disconnect any outstanding SF connections.

Prerequisite Calls

Connect for Structured Fields (120)

Call Parameters

	Standard Interface	Enhanced Interface
Function Number	Must be 121	
Data String	See the following table	
Length	Must be 3	Must be 8

	Standard Interface	Enhanced Interface
PS Position	NA	

Data String Contents

Byte		Definition
Standard	Enhanced	
1	1	A 1-character presentation space short name (PSID).
	2–4	Reserved.
2–3	5–6	Destination/origin unique ID returned by the Connect for structured field (120) functions.
	7–8	Reserved.

Return Parameters

Return Code	Explanation
0	The Disconnect from Structured Fields function was successful.
1	A specified host presentation space short session ID was not valid or was not connected.
2	An error was made in specifying parameters.
9	A system error occurred.
40	Disconnected with asynchronous requests pending.

Notes on Using This Function

1. When a **Disconnect from Structured Fields** function is called, any outstanding asynchronous **Read Structured Fields** (126) or **Write Structured Fields** (127) function requests are returned if the application issues the **Get Request Completion** (125) function call. Use the asynchronous form of this function when cleaning up after issuing a Disconnect call.
2. The **Reset System** (21) function will also free any outstanding asynchronous requests (requests that have not been retrieved by the application using the **Get Request Completion** (125) function).

Disconnect Presentation Space (2)

3270	5250	VT
Yes	Yes	Yes

The **Disconnect Presentation Space** function drops the connection between your EHLLAPI application program and the host presentation space. Also, if a host presentation space is reserved using the **Reserve** (11) function, it is released upon execution of the **Disconnect Presentation Space** function.

Prerequisite Calls

Connect Presentation Space (1)

Call Parameters

	Standard Interface	Enhanced Interface
Function Number	Must be 2	
Data String	NA	
Length	NA	
PS Position	NA	

Return Parameters

Return Code	Explanation
0	The Disconnect Presentation Space function was successful.
1	Your program was not currently connected to the host presentation space.
9	A system error was encountered.

Notes on Using This Function

1. After the **Disconnect Presentation Space** function is called, functions that interact with the host-connected presentation space are no longer valid (for example, the **Send Key** (3), **Wait** (4), **Reserve** (11) and **Release** (12) functions).
2. Your EHLLAPI application should disconnect from the host presentation space before exiting.
3. The **Disconnect Presentation Space** function does not reset the session parameters to the defaults. Your EHLLAPI application must call the **Reset System** (21) function to accomplish this.

Disconnect Window Service (102)

3270	5250	VT
Yes	Yes	Yes

The **Disconnect Window Service** function disconnects the window services connection between the EHLLAPI program and the specified host presentation space window.

Prerequisite Calls

Connect Window Services (101)

Call Parameters

	Standard Interface	Enhanced Interface
Function Number	Must be 102	
Data String	See the following table	
Length	1	4
PS Position	NA	

Data String Contents

Byte		Definition
Standard	Enhanced	
1	1	A 1-character presentation space short name (PSID)
	2–4	Reserved

Return Parameters

Return Code	Explanation
0	The Disconnect Window Service function was successful.
1	Your program is not connected for Window Services.
9	A system error occurred.

Notes on Using This Function

After the **Disconnect Window Service** function has been called, your application no longer manages the presentation space window.

Before exiting the application, you should request a **Disconnect Window Service** function for all presentation spaces that have been connected for Presentation Manager® services. If the application exits with an outstanding connection for window services, the subsystem cancels the outstanding connection.

EditKey Intercept

This feature enables you to intercept Edit keys in addition to the existing all keystrokes and send them to a session in a Windows 32-bit environment.

Prerequisites

1. Map the Edit functions in the Customize Keyboard window (for example Ctrl+C for edit copy function).
2. Call the Start Keystroke Intercept (50) EHLLAPI function with the call parameter data string value set. The values are as follows:

Byte Position	Contents
1	One of the following values: <ul style="list-style-type: none"> ◦ A specific host presentation space short name (PSID) ◦ A blank or null indicating a request for the host-connected host presentation space
2 to 4	Reserved
5	An option code character: <ul style="list-style-type: none"> ◦ D for AID keystrokes only ◦ L for all keystrokes ◦ E for all keystrokes and Edit keys ◦ M for requesting the asynchronous message mode of the notification (Windows only). If M is specified, a code character D or L, or E must be placed in position 13
6 to 8	Reserved
9 to 12	If M is specified in position 5, the window handle of the window that receives the message. The message is a non-zero return value of RegisterWindowMessage (PCSHLL).
13	If M is specified in position 5, one of the following values: <ul style="list-style-type: none"> ◦ D for AID keystrokes only ◦ L for all keystrokes ◦ E for all keystrokes and Edit keys
14 to 16	Reserved

3. To get the intercepted Edit keys, use the Get Key (51) EHLLAPI function. The key mnemonic returned in the data string for the Edit keys will have M (keystroke type mnemonic) at the 5th byte position. The next 4 bytes will have one of the following Edit key mnemonics based on the Edit key intercepted:

Key mnemonic	Key intercepted
@W@C	Edit Copy
@W@D	Edit Clear
@W@E	Edit Copy Append
@W@L	Edit Copy Link
@W@N	Edit Paste Next
@W@V	Edit Paste
@W@X	Edit Cut
@W@Z	Edit Undo

4. To send Edit keys to the session, use the Send Key (3) EHLLAPI function. The data string passed as the call parameter can specify the following Edit key mnemonics:

Key mnemonic	Key sent
@W@C	Edit Copy
@W@D	Edit Clear
@W@E	Edit Copy Append
@W@L	Edit Copy Link
@W@N	Edit Paste Next
@W@V	Edit Paste
@W@X	Edit Cut
@W@Z	Edit Undo

**Note:**

1. You do not have to call the Get Key (51) EHLLAPI function to use the Send Key (3) function. For both Get Key (51) and Send Key (3) functions to handle Edit keys, you must first call Start Keystroke Intercept (50) with the 5th byte position set to **E**. If the 5th byte contains **M**, then position 13 must contain **E**.
2. The expected return values for Start Keystroke Intercept (50), Get Key (51) and Send Key (3) functions have not changed.
3. Any prerequisites from the existing documentation should be followed as well as the prerequisites documented here.

Find Field Length (32)

3270	5250	VT
Yes	Yes	Yes

The **Find Field Length** function returns the length of a target field in the connected presentation space. This function can be used to find either protected or unprotected fields, but only in a *field-formatted* host presentation space.

This function returns the number of characters contained in the field identified using the call PS position parameter. This includes all characters from the beginning of the target field up to the character preceding the next attribute byte.

Prerequisite Calls

Connect Presentation Space (1)

Call Parameters

	Standard Interface	Enhanced Interface
Function Number	Must be 32	
Data String	See the following table	
Length	NA	NA
PS Position	See note	



Note: PS Position: Identifies the field within the host presentation space at which to start the **Find**. It can be the PS position of any byte within the field in which you desire the **Find** to start.

The calling 2-character data string can contain:

Code	Explanation
bb or Tb	This field
Pb	The previous field, either protected or unprotected.
Nb	The next field, either protected or unprotected
NP	The next protected field
NU	The next unprotected field
PP	The previous protected field
PU	The previous unprotected field



Note: The b symbol represents a required blank.

Return Parameters

This function returns a length and a return code.

Length:

The following lengths are valid:

Length	Explanation
= 0	When return code = 28, field length is 0. When return code = 24, host presentation space is not field formatted.
> 0	Required field length in the host presentation space.

Return Code:

The following codes are defined:

Return Code	Explanation
0	The Find Field Length function was successful.

Return Code	Explanation
1	Your program is not connected to a host session.
2	A parameter error was encountered.
7	The host presentation space position is not valid.
9	A system error was encountered.
24	No such field was found.
28	Field length of 0 bytes.

Notes on Using This Function

Except when `bb` or `Tb` is used as the calling data string, if the field found is the same as the field from which the **Find** started, a return code of 24 is returned.

Find Field Position (31)

3270	5250	VT
Yes	Yes	Yes

The **Find Field Position** function returns the beginning position of a target field in the host-connected presentation space. This function can be used to find either protected or unprotected fields but only in a *field-formatted* host presentation space.

Prerequisite Calls

Connect Presentation Space (1)

Call Parameters

	Standard Interface	Enhanced Interface
Function Number	Must be 31	
Data String	See the following table	
Length	NA	NA
PS Position	See note	



Note: PS Position: Identifies the field within the host presentation space at which to start the **Find**. It can be the PS position of any byte within the field in which you want the **Find** to start.

The calling 2-character data string can contain:

Code	Explanation
<code>bb</code> or <code>Tb</code>	This field
<code>Pb</code>	The previous field, either protected or unprotected

Code	Explanation
Nb	The next field, either protected or unprotected
NP	The next protected field
NU	The next unprotected field
PP	The previous protected field
PU	The previous unprotected field



Note: The `b` symbol represents a required blank.

Return Parameters

This function returns a length and a return code.

Length:

The following lengths are valid:

Length	Explanation
= 0	When return code = 28, field length is 0. When return code = 24, host presentation space is not field-formatted.
> 0	Relative position of the requested field from the origin of the host presentation space. This position is defined to be the first position after the attribute byte.

Return Code:

The following codes are defined:

Return Code	Explanation
0	The Find Field Position function was successful.
1	Your program is not connected to a host session.
2	A parameter error was encountered.
7	The host presentation space position is not valid.
9	A system error was encountered.
24	No such field was found.
28	Field length of 0 bytes.

Notes on Using This Function

Except when `bb` or `tb` is used as the calling data string, if the field found is the same as the field from which the **Find** started, a return code of 24 is returned.

Free Communications Buffer (124)

3270	5250	VT
Yes	No	No

The **Free Communications Buffer** function returns to management memory a buffer that is no longer required by the application. The application should free the buffer prior to exiting the system.

Prerequisite Calls

Allocate Communications Buffer (123)

Call Parameters

	Standard Interface	Enhanced Interface
Function Number	Must be 124	
Data String	See the following table	
Length	Must be 6	Must be 8
PS Position	NA	

Data String Contents

Byte		Definition
Standard	Enhanced	
1-2	1-4	Must be 0
3-6	5-8	The address of the buffer

Return Parameters

Return Code	Explanation
0	The Free Communications Buffer function was successful.
2	An error was made in specifying parameters.
9	A system error occurred.
41	The buffer is in use.

Notes on Using This Function

1. If the application attempts to free an in use buffer, the free request will be denied and a return code of 41 will be returned.
2. An application should request the **Free Communications Buffer** (124) function before exiting for all communication buffers that have been allocated using the **Allocate Communications Buffer** (123) function.
3. The **Reset System** (21) function will free buffers allocated by the **Allocate Communications Buffer** (123) function.

Get Key (51)

3270	5250	VT
Yes	Yes	Yes

The **Get Key** function lets your EHLLAPI application program retrieve a keystroke from a session specified by the **Start Keystroke Intercept** (50) function and either process, accept, or reject that keystroke. By placing this function in a loop, you can use it to intercept a string.

Prerequisite Calls

Start Keystroke Intercept (50)

Call Parameters

	Standard Interface	Enhanced Interface
Function Number	Must be 51	
Data String	See the following table	
Length	8	12
PS Position	NA	

Data String Contents

Byte		Definition
Standard	Enhanced	
1	1	One of the following values: <ul style="list-style-type: none"> • A 1-character presentation space short name (PSID) • A blank or null indicating a function call for the host-connected presentation
	2–4	Reserved

Byte		Definition
2–8	5–11	Blanks that hold space for the symbolic representation of the requested data
	12	Reserved

Return Parameters

This function returns a data string and a return code.

Data String:

See the following table:

Byte		Definition
Standard	Enhanced	
1	1	One of the following values: <ul style="list-style-type: none"> • A 1-character presentation space short name (PSID) • A blank or null indicating a function call for the host-connected presentation
	2–4	Reserved
2	5	An option code character, one of the following characters: <ul style="list-style-type: none"> • A for ASCII returned • M for keystroke mnemonic • S for special mnemonic
3–8	6–11	These 6 bytes of the preallocated buffer space are used internally to enqueue and dequeue keystrokes. Possible combinations include: <ul style="list-style-type: none"> • Byte 3 contains an ASCII character and byte 4 contains X'00' • Byte 3 contains the escape character (either @ or another character specified using the <code>ESC=c</code> option of function 9) and byte 4 contains a 1-byte abbreviation for a function. (See ASCII Mnemonics on page 20) • Bytes 5 through 8 might be similar to bytes 3 and 4 if the returned ASCII mnemonic is longer than 2 bytes (for example, if the ASCII mnemonic represents Attn @A@Q, byte 5 contains @ and byte 6 contains Q). If not used, bytes 5 through 8 are set to zero (X'00').

For clarification, some examples of returned data strings are provided below:



Note: The @ symbol is the default escape character. The value of the escape character can be set to any keystroke represented in ASCII by using the `ESC=c` option of the **Set Session Parameters** (9) function. If the escape character has been changed to another character using this option, the @ symbol in the following examples is replaced by the other character.

16-Bit Interface

`EAt`

`E` is the presentation space short name. The keystrokes are returned as ASCII (A), and the returned key is the lowercase letter t. (Bytes 4–8 = X'00').

`EM@2`

`E` is the presentation space short name. The keystrokes are returned as mnemonics, and the returned key is PF2 (Bytes 5–8 = X'00').

32-Bit Interface

`EbbbAt`

`E` is the presentation space short name. The keystrokes are returned as ASCII (A), and the returned key is the lowercase letter t. (Bytes 7–11 = X'00').

`EbbbM@2`

`E` is the presentation space short name. The keystrokes are returned as mnemonics, and the returned key is PF2 (Bytes 8–11 = X'00').

Return Code:

The following codes are valid:

Return Code	Explanation
0	The Get Key function was successful.
1	An incorrect presentation space was specified.
5	You specified the AID only option under the Start Keystroke Intercept (50) function, and non-AID keys are inhibited by this session type when EHLLAPI tries to write incorrect keys to the presentation space.
8	No prior Start Keystroke Intercept (50) function was called for this presentation space.
9	A system error was encountered.
20	An undefined key combination was typed.
25	The requested keystrokes are not available on the input queue.
31	Keystroke queue overflowed and keystrokes were lost.

Notes on Using This Function

1. If a return code of 31 occurs for the **Get Key** function, either:
 - Increase the value of the calling length parameter for the **Start Keystroke Intercept** (50) function, or
 - Execute the **Get Key** function more frequently.

An intercepted keystroke occupies 3 bytes in the buffer. The next intercepted keystroke is placed in the adjacent three bytes. When the **Get Key** function retrieves a keystroke (first in first out, FIFO), the three bytes that it occupied are made available for another keystroke. By increasing the size of the buffer or the rate at which keystrokes are retrieved from the buffer, you can eliminate buffer overflow.

For the PC/3270, another way to eliminate return code 31 is to operate the PC/3270 emulator in the resume mode.

2. You can use the **Send Key** (3) function to pass both original keystrokes and any others that your EHLLAPI application might need to the host-connected presentation space.
3. Keystrokes arrive asynchronously and are enqueued in the keystroke queue that you have provided in your EHLLAPI application program using the **Start Keystroke Intercept** (50) function.
4. The **Get Key** function behaves like a read. When keystrokes are available, they are read into the data area that you have provided in your application.
5. In the case of field support for a session, the application might be interested only in AID keys, for example the Enter key. If so, the **Start Keystroke Intercept** (50) function option code should be set to **D** (meaning for AID Keys only).
6. To use this function, preallocate memory to receive the returned data string parameter. The statements required to preallocate this memory vary depending on the language in which your application is written. Refer to [Memory Allocation on page 10](#) for more information.

Get Request Completion (125)

3270	5250	VT
Yes	No	No

The **Get Request Completion** function allows an application to determine the status of a previous asynchronous function request issued to the EHLLAPI and to obtain the function parameter list before using the data string again. This function is valid only if the user specified asynchronous (A) completion on a previous function call such as **Read Structured Fields** (126) or **Write Structured Fields** (127).

Each asynchronous request requiring the **Get Request Completion** function will return a unique ID from the asynchronous request. The application must save this ID. This ID is the identification used by the **Get Request Completion** function to identify the desired request. The user has three request options using this function:

1. The application can query or wait for a specific asynchronous function request by supplying the request ID of that function and a nonblank session short name.
2. The application can query or wait for the first completed asynchronous function request for a specified session by supplying a request ID of X'0000' and a nonblank session short name.

Prerequisite Calls

Connect Structured Fields (120) and **Allocate Communications Buffer** (123)

and

Read Structured Fields (126) or **Write Structured Fields** (127)

Call Parameters

	Standard Interface	Enhanced Interface
Function Number	Must be 125	
Data String	See the following table	
Length	Must be 14	Must be 24
PS Position	NA	

Data String Contents

Byte		Definition
Standard	Enhanced	
1	1	A 1-character presentation space short name (PSID)
	2–4	Reserved
2	5	N or W N=NOWAIT is required W=WAIT is required
	6–8	Reserved
3–4	9–10	Function request ID.
5–6	11–12	Reserved
7–10	13–16	Reserved
11–12	17–20	Reserved
13–14	21–24	Reserved

The **Get Request Completion** function behaves differently depending upon the second character of the parameter string, which is one of the following characters:

N

Nowait option: If a specific request ID was supplied and the function has completed, control will be returned to the application with a return code of zero and a completed data string as defined in [Return Parameters on page 90](#). If a request ID of zero was supplied and any eligible asynchronous function

has completed, control will be returned to the application with a return code of zero and a completed data string as defined in [Return Parameters on page 90](#).

W

Wait option: If a specific request ID was supplied and the function has not completed, the call will wait until the function has completed before returning to the application. If the supplied request ID was zero and no eligible asynchronous function has completed, the call will wait until a function completes before returning to the calling application. On return, the return code value will be zero and the data string will be completed as defined in [Return Parameters on page 90](#).

Return Parameters

Byte		Definition
Standard	Enhanced	
5–6	11–12	Function number of the completed asynchronous function (126 or 127). (returned)
7–10	13–16	Address of the data string of the completed asynchronous function call. (The application must not reuse the data string until the request has completed). (returned)
11–12	17–20	Length of the data string of the completed asynchronous function call. (returned)
13–14	21–24	Return code of the completed asynchronous function call. (returned)

Return Code	Explanation
0	The Get Request Completion function was successful.
2	An error was made in specifying parameters.
9	A system error was encountered.
38	Requested function was not complete.
42	No matching request was found.

There are some differences between return codes 38 and 42:

1. Return code 38
 - a. If a specific request ID and session were requested, both the session and ID were found but the request is pending (not in a completed state).
 - b. If a zero request ID and a specific session were requested, the specified session has pending requests, but they are not satisfied (complete).
 - c. If a zero request ID and a blank session were requested, pending requests were found but none were satisfied (complete).
2. Return code 42

- a. If a specific request ID and session were requested, the specific request ID was not found in either a pending or a completed state.
- b. If a zero request ID and a specific session were requested, the specific session contains no pending or completed requests.
- c. If a zero request ID and a blank session were requested, no pending or completed requests were found.

Notes on Using This Function

1. This function is valid only if the user specified asynchronous completion (A for Asynchronous) on a previous function call such as **Read Structured Fields** or **Write Structured Fields**.
2. If the return code is a 0, the application should check the returned data string for information pertaining to the completion of the requested asynchronous function.

Lock Presentation Space API (60)

3270	5250	VT
Yes	No	No

The **Lock Presentation Space API** function allows the application to obtain or release exclusive control of the presentation space window over other Windows 32-bit applications. While locked, no other application can connect to the presentation space window.

Successful processing of this function with the Lock causes EHLLAPI presentation space window functions requested from other EHLLAPI applications to be queued until the requesting application unlocks the presentation space. Requests from the locking application are processed normally.

Prerequisite Calls

Connect to Presentation Space (1)

Call Parameters

	Standard Interface	Enhanced Interface
Function Number	Must be 60	
Data String	See the following table	
Length	Must be 3	Must be 8
PS Position	NA	

Data String Contents

Byte		Definition
Standard	Enhanced	
1	1	A 1-character presentation space short name (PSID).
	2–4	Reserved.
2	5	One of the following characters: <ul style="list-style-type: none"> • L to lock the API. • U to unlock the API.
3	6	One of the following characters: <ul style="list-style-type: none"> • R to return if the presentation space is already locked by an application. • Q to queue the Lock request if the presentation space is already locked by an application.
	7–8	Reserved.

Return Parameters

Return Code	Explanation
0	The Lock Presentation Space API function was successful.
1	An incorrect host presentation space short session ID was specified or was not connected.
2	An error was made in specifying parameters.
9	A system error was encountered.
43	The API was already locked by another EHLLAPI application (on LOCK) or API not locked (on UNLOCK).

Notes on Using This Function

The following EHLLAPI functions are queued when a lock is in effect:

- **Send Key** (3)
- **Copy Presentation Space** (5)
- **Search Presentation Space** (6)
- **Copy Presentation Space to String** (8)
- **Release** (11)
- **Reserve** (12)
- **Query Field Attribute** (14)

- **Copy String to Presentation Space** (15)
- **Search Field** (30)
- **Find Field Position** (31)
- **Find Field Length** (32)
- **Copy String to Field** (33)
- **Copy Field to String** (34)
- **Set Cursor** (40)
- **Send File** (90)
- **Copy Presentation Space to Clipboard** (35)
- **Paste Clipboard to Presentation Space** (36)
- **Receive File** (91)
- **Connect to Presentation Space** (1) with the CONPHYS parameter set in a previous **Set Sessions Parameter** (9) function call.

These queued requests are not serviced until the lock is removed. When the lock is removed, the queued requests are processed in first-in-first-out (FIFO) order. EHLLAPI functions not listed are run as if there was no lock. The requesting application unlocks the presentation space window by one of the following methods:

- Disconnecting from the presentation space while still owning the Lock.
- Issuing the **Reset System** (21) function while still owning the Lock.
- Stopping the application while still owning the Lock.
- Stopping the session.
- Successfully issuing the **Lock Presentation Space API** with the Unlock option.

Before exiting the application, you should unlock any presentation space windows that have been locked with the **Lock Presentation Space API** function. If the application exits with outstanding locks, or a **Reset System** (21), or **Disconnect Presentation Space** (2) function is issued, the locks are released.

It is recommended that applications lock the presentation space only for short periods of time and only when exclusive use of the presentation space is required.

Lock Window Services API (61)

3270	5250	VT
Yes	No	No

The **Lock Window Services API** function allows the application to obtain or release exclusive control of the presentation space window over other Windows 32-bit applications. While locked, no other application can connect to the presentation space window.

Successful processing of this function with the Lock causes EHLLAPI presentation space window functions requested from other EHLLAPI applications to be queued until the requesting application unlocks the presentation space. Requests from the locking application are processed normally.

Prerequisite Calls

Connect Window Services (101)

Call Parameters

	Standard Interface	Enhanced Interface
Function Number	Must be 61	
Data String	See the following table.	
Length	Must be 3	Must be 8
PS Position	NA	

Data String Contents

Byte		Definition
Standard	Enhanced	
1	1	A 1-character presentation space short name (PSID).
	2–4	Reserved.
2	5	One of the following characters: <ul style="list-style-type: none"> • L to lock the API. • U to unlock the API.
3	6	One of the following characters: <ul style="list-style-type: none"> • R to return if the presentation space is already locked by an application. • Q to queue the Lock request if the presentation space is already locked by an application.
5–6	11–12	Function number of the completed asynchronous function (126 or 127). (returned)
	7–8	Reserved.

Return Parameters

Return Code	Explanation
0	The Lock Window Services API function was successful.
1	An incorrect host presentation space short session ID was specified or was not connected.
2	An error was made in specifying parameters.
9	A system error was encountered.

Return Code	Explanation
38	Requested function was not complete.
43	The API was already locked by another EHLLAPI application (on LOCK) or API not locked (on UNLOCK).

Notes on Using This Function

The following EHLLAPI functions are queued when a lock is in effect:

- **Window Status** (104)
- **Change Switch List Name** (105)
- **Change PS Window Name** (106)

These queued requests are not serviced until the lock is removed. When the lock is removed, the queued requests are processed in first-in-first-out (FIFO) order.

The requesting application unlocks the presentation space window by one of the following methods:

- Successfully issuing the **Lock Window Services API** with the UNLOCK option.
- Disconnecting from the presentation space while still owning the Lock.
- Issuing the **Reset System** (21) function while still owning the Lock.
- Stopping the application while still owning the Lock.
- Stopping the session.

Before exiting the application, you should Unlock any presentation space windows that have been locked with the **Lock Window Services API** function. If the application exits with outstanding locks, the subsystem releases the locks.

It is recommended that applications lock the presentation space only for short periods of time and only when exclusive use of the presentation space is required.

Pause (18)

3270	5250	VT
Yes	Yes	Yes

The **Pause** function waits for a specified amount of time. It should be used in place of *timing loops* to wait for an event to occur. A **Pause** function can be ended by a host event if a prior **Start Host Notification** (23) function has been called and the IPAUSE option is selected.

Prerequisite Calls

There are no prerequisite calls for this function.

Call Parameters

	Standard Interface	Enhanced Interface
Function Number	Must be 18	
Data String	NA	
Length	Contains the pause duration in half-second increments	
PS Position	NA	

Return Parameters

Return Code	Definition
0	The wait duration has expired.
9	An internal system error was encountered. The time results are unpredictable.
26	The host session presentation space or OIA has been updated. Use the Query Host Update (24) function to get more information.

Notes on Using This Function

1. Selecting the FPAUSE or IPAUSE option using the **Set Session Parameters** (9) function affects the length of the pause you get when you call this function. See item [6 on page 142](#) for more information.
2. The value entered in the calling length parameter is the maximum number of half-second intervals that the **Pause** function waits. For a pause of 20 seconds, a hex value of `0028` (decimal 40) must be passed in the calling length parameter.
3. If you use the IPAUSE option and the pause value is zero, then the function waits up to 2400 half-second intervals, unless interrupted sooner. If you use the FPAUSE option and the pause value is zero, then the function returns immediately.
4. If you use the IPAUSE option, once a pause has been satisfied by a host event, you should call the **Query Host Update** (24) function to clear the queue prior to the next **Pause** function. The **Pause** function will continue to be satisfied with the pending event until the **Query Host Update** (24) function is completed.
5. A practical maximum value for the **Pause** function is `2400`. You should not use the **Pause** function for these kinds of tasks:
 - Delay for very long durations (of several hours, for example).
 - Delay for more than a moderate length of time (20 minutes) before checking the system time-of-day clock and proceeding with your EHLLAPI program execution.
 - With applications requiring a high-resolution timer because the time interval created by a **Pause** function is approximate.
 - Set the time interval to zero in a loop.
6. IPAUSE set and the interruptible pause allow an EHLLAPI application to determine whether the specified host presentation space (PS) or operator information area (OIA) is updated. The following three functions are used:

- **Start Host Notification** (23)
- **Query Host Update** (24)
- **Stop Host Notification** (25)

By using IPAUSE when the **Start** function is called, you can make an application wait until the host presentation space or OIA (or both) receives an update. When the receive is completed and the application can issue the **Query** function to determine the changes, **Pause** terminates. Then the application issues the **Search Presentation Space** (6) to check whether the expected update occurred.

Post Intercept Status (52)

3270	5250	VT
Yes	Yes	Yes

The **Post Intercept Status** function informs the Z and I Emulator for Windows emulator that a keystroke obtained through the **Get Key** (51) function was accepted or rejected. When the application rejects a keystroke, the **Post Intercept Status** function issues a beep.

Prerequisite Calls

Start Keystroke Intercept (50)

Call Parameters

	Standard Interface	Enhanced Interface
Function Number	Must be 52	
Data String	See the following table	
Length	Must be 2	Must be 8
PS Position	NA	

The calling data string can contain:

Byte		Definition
Standard	Enhanced	
1	1	One of the following values: <ul style="list-style-type: none"> • The 1-letter short name of the presentation space. • A blank or null indicating a function call for the host-connected presentation space.
	2–4	Reserved
2	5	One of the following characters:

Byte		Definition
		<ul style="list-style-type: none"> • A for accepted keystroke. • R for rejected keystroke.
	6–8	Reserved.

Return Parameters

Return Code	Explanation
0	The Post Intercept Status function was successful.
1	An incorrect presentation space was specified.
2	An incorrect session option was specified.
8	No prior Start Keystroke Intercept (50) function was called for this presentation space ID.
9	A system error was encountered.

Query Additional Field Attribute (45)

3270	5250	VT
No	Yes	No

The **Query Additional Field Attribute** function returns additional information about the 5250 field containing the input host presentation space position. This information is returned in the data string parameter in the form of a defined structure.

Prerequisite Calls

Connect Presentation Space (1)

Call Parameters

	Standard Interface	Enhanced Interface
Function Number	Must be 45.	
Data String	8 bytes long character string.	
Length	8 is implied.	
PS Position	Identifies the target. This can be the PS position of any byte within the target field.	

The calling data string can contain:

Byte	Definition
1–8	Reserved

Return Parameters

This function returns a data string and a return code.

Data String:

The function returns the following data string.

Byte	Definition
1–6	Reserved
7–8	Two 8–bit unsigned characters that return: <ul style="list-style-type: none"> • R if field is RTL and L if field is LTR. • U if field is upper case and L if field is a normal case field.

Return Code:

The following return codes are defined:

Return Code	Explanation
0	The Query Additional Field Attribute was successful.
1	Your program is not currently connected to a host session.
7	The host presentation space position is not valid.
9	No field was found in this position.
24	Field is unformatted.

Query Close Intercept (42)

3270	5250	VT
Yes	Yes	Yes

The **Query Close Intercept** function allows the application to determine if the close option was selected.

Prerequisite Calls

Start Close Intercept (41)

Call Parameters

	Standard Interface	Enhanced Interface
Function Number	Must be 42	

	Standard Interface	Enhanced Interface
Data String	See the following table.	
Length	Must be 1	Must be 4
PS Position	NA	

The calling data string can contain:

Byte		Definition
Standard	Enhanced	
1	1	1-character short session ID of the host presentation space, or a blank or null indicating request for querying the host-connected session
	2–4	Reserved

Return Parameters

Return Code	Explanation
0	A close intercept event did not occur.
1	The presentation source was not valid.
2	An error was made in specifying parameters.
8	No prior Start Close Intercept (41) function was called for this host presentation space.
9	A system error occurred.
12	The session stopped.
26	A close intercept occurred since the last query close intercept call.

Query Communications Buffer Size (122)

3270	5250	VT
Yes	No	No

The **Query Communications Buffer Size** function allows an application to determine both the maximum and the optimum buffer sizes supported by the emulation program.

Prerequisite Calls

There are no prerequisite calls for this function.

Call Parameters

	Standard Interface	Enhanced Interface
Function Number	Must be 122	
Data String	See the following table	
Length	Must be 9	Must be 20

	Standard Interface	Enhanced Interface
PS Position	NA	

The calling data string can contain:

Byte		Definition
Standard	Enhanced	
1	1	A 1-character presentation space short name (PSID)
	2–4	Reserved
2–3	5–8	16- or 32-bit field for the optimum supported inbound buffer size (Returned value)
4–5	9–12	16- or 32-bit field for the maximum supported inbound buffer size (Returned value)
6–7	13–16	16- or 32-bit field for the optimum supported outbound buffer size (Returned value)
8–9	17–20	16- or 32-bit field for the maximum supported outbound buffer size (Returned value)

Return Parameters

Return Code	Explanation
0	The Query Communications Buffer Size function was successful.
1	A specified host presentation space short session ID was not valid or was not connected.
2	An error was made in specifying parameters.
9	A system error occurred.
10	The function was not supported by the emulation program.

Notes on Using This Function

1. There is no way to require the user to use this function. It is not a required function so that the application can be tailored to run on any system.
2. The buffer sizes returned represent the record sizes that are actually transmitted across the medium. For a DDM connection, the 8-byte header supplied in the **Read** and **Write Structured Fields** data buffer is stripped off and 1 byte containing the structured field AID value is prefixed. The application should compare the size of the actual data in the data buffer (which does not include the 8-byte header) with the buffer sizes returned by the **Query Communications Buffer Size** minus 1 byte. For destination/origin connections, the 8-byte header supplied in the **Read** and **Write Structured Fields** data buffer is stripped off and 9 bytes are then prefixed to the data. The application should compare the size of the actual data in the data buffer (which does not include the 8-byte header) with the buffer size returned by the **Query Communications Buffer Size** minus 9 bytes.

3. The maximum buffer sizes returned represent the maximum number of bytes supported by the workstation hardware and by the emulator. The maximum buffer size can be used only if the host is also configured to accept at least these maximum sizes.
4. The optimum buffer sizes returned represent the optimum number of bytes supported by the both the workstation hardware and the emulator. Some network configurations might set transmission limits smaller than these values. In these cases, the data transfer buffer size override value in the emulator configuration profile will be used for structured field support. The **Query Communications Buffer Size** will reflect any buffer size override values entered in the emulator configuration profile.

Query Communication Event (81)

3270	5250	VT
Yes	Yes	Yes

The **Query Communication Event** function lets the EHLLAPI program determine whether any communication events have occurred.

Prerequisite Calls

Start Communication Notification (80)

Call Parameters

	Enhanced Interface
Function Number	Must be 81
Data String	1-character short name of the host presentation space or a blank or null indicating request for updates to the host-connected presentation space
Length	4 is implied
PS Position	NA

The calling data structure contains these elements:

Byte	Definition
1	A 1-character presentation space short name (PSID)
2-4	Reserved

Return Parameters

Return Code	Definition
0	The function was successful
1	An incorrect PSID was specified
8	No prior call to Start Communication Notification (80) function was called for the PSID
9	A system error was encountered

21	The indicated PSID was connected
22	The Indicated PSID was disconnected

Query Cursor Location (7)

3270	5250	VT
Yes	Yes	Yes

The **Query Cursor Location** function indicates the position of the cursor in the host-connected presentation space by returning the cursor position.

Prerequisite Calls

Connect Presentation Space (1)

Call Parameters

	Standard Interface	Enhanced Interface
Function Number	Must be 7	
Data String	NA	
Length	NA	
PS Position	NA	

Return Parameters

This function returns a length and a return code.

Length:

Host presentation space position of the cursor.

Return Code:

The following codes are defined:

Return Code	Explanation
0	The Query Cursor Location function was successful.
1	Your program is not currently connected to a host session.
9	A system error was encountered.

Query Field Attribute (14)

3270	5250	VT
Yes	Yes	Yes

The **Query Field Attribute** function returns the attribute byte of the field containing the input host presentation space position. This information is returned in the returned length parameter.

For the PC/3270, note also that:

- The returned length parameter is set to 0 if the screen is unformatted.
- Attribute bytes are equal to or greater than hex C0.

Prerequisite Calls

Connect Presentation Space (1)

Call Parameters

	Standard Interface	Enhanced Interface
Function Number	Must be 14.	
Data String	NA.	
Length	NA.	
PS Position	Identifies the target. This can be the PS position of any byte within the target field.	

Return Parameters

This function returns a length and a return code.

Length:

The attribute value if the screen is formatted, or 0 if the screen is unformatted.

Return Code:

The following codes are defined:

Return Code	Explanation
0	The Query Field Attribute was successful.
1	Your program is not currently connected to a host session.
7	The host presentation space position is not valid.
9	A system error was encountered.
24	Attribute byte not found or unformatted host presentation space.

Notes on Using This Function

The returned field attributes are defined in the following tables. The bit positions are in IBM format with bit 0 as the left most bit in the byte.

• 3270 field attribute:

Bit Position	Meaning
0–1	Both = 1, field attribute byte
2	Unprotected/protected 0 = Unprotected data field 1 = Protected field
3	A/N 0 = Alphanumeric data 1 = Numeric data only
4–5	I/SPD 00 = Normal intensity, pen not detectable 01 = Normal intensity, pen detectable 10 = High intensity, pen detectable 11 = Nondisplay, pen not detectable
6	Reserved
7	MDT 0 = Field has not been modified 1 = Field has been modified

• 5250 field attributes:

Bit Position	Meaning
0	Field attribute flag 0 = Nonfield attribute flag 1 = Field attribute flag
1	Visibility 0 = Nondisplay 1 = Display
2	Unprotected/protected

Bit Position	Meaning
	0 = Unprotected data field 1 = Protected field
3	Intensity 0 = Normal intensity 1 = High intensity
4–6	Field type 000 = Alphanumeric data: All characters are available 001 = Alphabet only: Uppercase and lowercase, comma, period, hyphen, blank, or Dup key are available 010 = Numeric shift: Automatic shift for number 011 = Numeric data only: 0–9, comma, period, plus, minus, blank, or Dup key are available 101 = Numeric data only: 0–9, or Dup key are available 110 = Magnetic stripe reading device data only 111 = Signed-numeric data: 0–9, plus, minus, or Dup key are available
7	MDT 0 = Field has not been modified 1 = Field has been modified

Query Host Update (24)

3270	5250	VT
Yes	Yes	Yes

The **Query Host Update** function lets the programmed operator determine if the host has updated the host presentation space or OIA because:

- The **Start Host Notification** (23) function was called (on first call to the **Query Host Update** function only)
- The previous call to the **Query Host Update** function (for all calls to the **Query Host Update** function except the first).

Prerequisite Calls

Start Host Notification (23)

Call Parameters

	Standard Interface	Enhanced Interface
Function Number	Must be 24	
Data String	1-character short name of the host presentation space, or a blank or null indicating request for updates to host-connected presentation space	
Length	1 is implied	4 is implied
PS Position	NA	

The calling data string can contain:

Byte		Definition
Standard	Enhanced	
1	1	A 1-character presentation space short name (PSID)
	2–4	Reserved

Return Parameters

Return Code	Definition
0	No updates have been made since the last call.
1	An incorrect host presentation space was specified.
8	No prior Start Host Notification (23) function was called for the host presentation space ID.
9	A system error was encountered.
21	The OIA was updated.
22	The presentation space was updated.
23	Both the OIA and the host presentation space were updated.
44	Printing has completed in the printer session.

Notes on Using This Function

The target presentation space must be specified in the data string, even though a connection to the host presentation space is not necessary to check for updates.

Query Session Status (22)

3270	5250	VT
Yes	Yes	Yes

The **Query Session Status** function is used to obtain session-specific information.

Prerequisite Calls

There are no prerequisite calls for this function.

Call Parameters

	16-bit	32-bit
Function Number	Must be 22.	
Data String	An 18/20-byte string consisting of a 1-byte short name of the target presentation space plus 17 bytes for returned data. Position 1 can be filled with: <ol style="list-style-type: none"> 1. A blank or a null to indicate a request for the host_connected presentation space. 2. An * (asterisk) to indicate a request for the keyboard-owner presentation space. 	
Length	Must be 18	Must be 20
PS Position	NA	

Return Parameters

This function returns a data string and a return code.

Byte		Definition
Standard	Enhanced	
1	1	A 1-character presentation space short name (PSID)
	2–4	Reserved
2–9	5–12	Session long name (same as profile name; or, if profile not set, same as short name)
10	13	Session Type D 3270 display E 3270 printer F 5250 display G 5250 printer

Byte		Definition
		H ASCII VT
11	14	Session characteristics expressed by a binary number including the following session-characteristics bits Bit 0 EAB 0: Session has the basic attribute. 1: Session has the extended attribute Bit 1 PSS 0: Session does not support the programmed symbols 1: Session supports the programmed symbols Bits 2–7 Reserved
12–13	15–16	Number of rows in the host presentation space, expressed as a binary number
14–15	17–18	Number of columns in the host presentation space, expressed as a binary number
16–17	19–20	Host code page expressed as a binary number
18		Reserved

Return Code:

The following codes are defined:

Return Code	Explanation
0	The Query Session Status function was successful.
1	An incorrect host presentation space was specified.
2	An incorrect string length was made.
9	A system error was encountered.

Notes on Using This Function

1. To use this function, preallocate memory to receive the returned data string parameter. The statements required to preallocate this memory vary depending on the language in which your application is written. See [Memory Allocation on page 10](#) for more information.

Query Sessions (10)

3270	5250	VT
Yes	Yes	Yes

The **Query Sessions** function returns a 16-byte (12-byte for standard interface) data string describing each host session.

Prerequisite Calls

There are no prerequisite calls for this function.

Call Parameters

Function	Description	
	Standard Interface	Enhanced Interface
Function Number	Must be 10	
Data String	Preallocated string of $16n$ bytes long ($12n$ for 16-bit) (n = number of sessions) or more	
Length	$12n$ bytes	$16n$ bytes
PS Position	NA	



Note: When the length is not matched to the number of sessions, the return code is 2.

Return Parameters

This function returns a data string, a length, and a return code.

Data String:

The returned data string is $16n$ bytes long ($12n$ for standard interface), where n is the number of host sessions. The descriptors are concatenated into the data string and each session type, and presentation space size of a host session.

The format of each 16-byte (12-byte for standard interface) session descriptor is as follows:

Byte		Definition
Standard	Enhanced	
1	1	A 1-character presentation space short name (PSID)
	2–4	Reserved
2–9	5–12	Session long name (same as profile name; or, if profile not set, same as short name)
10	13	Connection type H=host
	14	Reserved
11–12	15–16	Host presentation space size (this is a binary number and is not in display format). If the session type is a print session, the value is 0.

Length:

The number of host sessions started.

Return Code:

The following codes are defined:

Return Code	Explanation
0	The Query Sessions function was successful.
2	An incorrect string length was made.
9	A system error was encountered.

Notes on Using This Function

1. If an application program receives RC=2 or RC=0, the number of the active sessions is returned in the length field. The application program can recognize the minimum string length by this number.
2. The **Query Sessions** function is affected by the CFGSIZE/NOCFGZISE session option (see item 16 on page 146 for more information) and by the EXTEND_PS/NOEXTEND_PS option (see item 20 on page 147 for more information).

**Note:**

1. When NOCFGSIZE is set in **Set Session Parameters** (9) for a 5250 session, the value of presentation space size returned in byte position 11 and 12 from **Query Sessions**(10) will be changed in accordance with the selection of EXTEND_PS or NOEXTEND_PS.
2. When EXTEND_PS is set in **Set Session Parameters** (9), presentation space size returned from **Query Sessions** (10) will include the size of the message line, if it exists.
3. When NOEXTEND_PS is set, the value will not change regardless of the existence of a message line. In the case of 25 row, 80 column presentation space, the value can be 1920 or 2000.

Query System (20)

3270	5250	VT
Yes	Yes	Yes

The **Query System** function can be used by an EHLLAPI application program to determine the level of Z and I Emulator for Windows support and other system-related values. This function returns a string that contains the appropriate system data. Most of this information is for use by a service coordinator when you call the IBM Support Center after receiving a return code 9 (a system error was encountered).

The bytes in this returned string are defined in [Return Parameters on page 112](#).

Prerequisite Calls

There are no prerequisite calls for this function.

Call Parameters

	Standard Interface	Enhanced Interface
Function Number	Must be 20	
Data String	Preallocated string of 35 bytes	36 bytes
Length	Must be 35	Must be 36
PS Position	NA	

Return Parameters

This function returns a data string and a return code.

Data String:

A data string of 35 bytes (for 16-bit) or 36 bytes (for 32-bit) is returned. The bytes are defined as follows:

Byte		Definition
Standard	Enhanced	
1	1	EHLLAPI version number
2-3	2-3	EHLLAPI level number
4-9	4-9	Reserved
10-12	10-12	Reserved
13	13	Hardware base, U=Unable to determine
14	14	Program type, where P=HCL Z and I Emulator for Windows
15-16	15-16	Reserved
17-18	17-18	Z and I Emulator for Windows version/level as a 2-byte ASCII value
19	19	Reserved
20-23	20-23	Reserved
24-27	24-27	Reserved
28-29	28-29	Reserved
	30	Reserved
30-31	31-32	NLS type expressed as a 2-byte binary number
33-35	34-36	Reserved

Return Code

The following codes are defined:

Return Code	Explanation
0	The Query System function was successful; data string has been returned.
1	EHLLAPI is not loaded. (PC/3270 only)
2	An incorrect string length was specified. (PC/3270 only)
9	A system error was encountered.

Notes on Using This Function

To use this function, preallocate memory to receive the returned data string parameter. See [Memory Allocation on page 10](#) for more information.

Query Window Coordinates (103)

3270	5250	VT
Yes	Yes	Yes

The **Query Window Coordinates** function requests the coordinates for the window of a presentation space. The window coordinates are returned in pels.



Note: (0,0) indicates the top-left of the window.

Prerequisite Calls

Connect Window Services (101)

Call Parameters

	Standard Interface	Enhanced Interface
Function Number	Must be 103	
Data String	1-character short session ID of the host presentation space	
Length	17 is implied	20 is implied
PS Position	NA	

The calling data string can contain:

Byte		Definition
Standard	Enhanced	
1	1	One of the following values:

Byte		Definition
		<ul style="list-style-type: none"> • A 1-character presentation space short name (PSID) • A blank or null indicating a function call for the current connection presentation space
	2–4	Reserved
2–17	5–20	Reserved

Return Parameters

This function returns a data string and a return code.

Byte		Definition
Standard	Enhanced	
1	1	One of the following values: <ul style="list-style-type: none"> • A 1-character presentation space short session ID • A blank or null indicating a function call for the current connection presentation space
	2–4	Reserved
2–17	5–20	Four 32-bit unsigned integers that return:
2–5	5–8	XLeft Long integer in pels of the left X coordinate of the rectangular window relative to the desktop window
6–9	9–12	YBottom Long integer in pels of the bottom Y coordinate of the rectangular window relative to the desktop window
10–13	13–15	XRight Long integer in pels of the right X coordinate of the rectangular window relative to the desktop window
14–17	16–20	YTop Long integer in pels of the top Y coordinate of the rectangular window relative to the desktop window

Return Code:

The following codes are defined:

Return Code	Explanation
0	The Query Window Coordinates function was successful.
1	Your program was not currently connected to the host session.
9	A system error occurred.
12	The session stopped.

Read Structured Fields (126)

3270	5250	VT
Yes	No	No

The **Read Structured Fields** function allows an application to read structured field data from the host application. If the call specifies S (for Synchronous), the application does not receive control until the **Read Structured Fields** is completed. If the call specifies A (for Asynchronous), the application receives control immediately after the call. If the call specifies M (for Asynchronous, message mode), the application receives control immediately after the call. The application can wait for the message. In any case (S, A, or M), the application provides the buffer address in which the data from the host is to be placed.

For a successful asynchronous completion of this function, the following statements apply:

The return code field in the parameter list might not contain the results of the requested I/O. If the return code is not 0, the request failed. The application must take the appropriate action based on the return code.

If the return code for this request is 0, the application must use the request ID returned with this function call to issue the **Get Request Completion** function call to determine the completion results of the function associated with the request ID. The **Get Request Completion** function call returns the following information:

1. Function request ID
2. Address of the data string from the asynchronous request
3. Length of the data string
4. Return code of the completed function

Prerequisite Calls

Connect for Structured Fields (120) and **Allocate Communication Buffer** (123)

Call Parameters

	Standard Interface	Enhanced Interface
Function Number	Must be 126	
Data String	See the following table	
Length	8, 10 or 14	20
PS Position	NA	

The calling data string can contain:

Byte		Definition
Standard	Enhanced	
1	1	A 1-character presentation space short name (PSID).

Byte		Definition
	2–4	Reserved.
2	5	S or A or M S = Synchronous. Control is not returned to the application until the read is satisfied. A = Asynchronous. Control is returned immediately to the application, can wait for the event object. M = Asynchronous. Control is returned immediately to the application, can wait for the message.
	6	Reserved.
3–4	7–8	2-byte destination/origin ID.
5–8	9–12	4-byte address of the buffer into which the data is to be read. The buffer must be obtained using the Allocate Communications Buffer (123) function.
9–10	13–16	Reserved.
11–12	17–20	When M is specified in position 2 the window handle of the window that receives the message should be set. The message is a return value of RegisterWindowMessage ("PCSHLL")(not equal 0).
13–14		The data in these positions is ignored by EHLLAPI. However, no error is caused if the migrating program has data in these positions. This data is accepted to provide compatibility with migrating applications.

Return Parameters

This function returns a data string and a return code.

Data String:

If A (asynchronous) is specified in position 5, (2 for standard interface) and the function is completed successfully, the following data string is returned:

Byte		Definition
Standard	Enhanced	
9–10	13–14	2-byte function request ID. It is used by the Get Request Completion (125) function to determine the completion of this function call.
	15–16	Reserved.
	17–20	4-byte value in which the event object address is returned by EHLLAPI. The application can wait for this event object. When the event object is cleared,

Byte		Definition
		the application must issue the Get Request Completion (125) function call (32-bit only).



Note: A event object address is returned for each successful asynchronous request. The event object should not be used again. A new event object is returned for each request and is valid for only the duration of that request.

Data String:

If "M" (asynchronous message mode) is specified in position 5 (2 for 16-bit applications) and the function is completed successfully, the following data string is returned:

Byte		Definition
9–10	13–14	A 2-byte function request ID. It is used by the Get Request Completion (125) function to determine the completion of this function call.
	15–16	Reserved.
11–12	17–18	Task ID of asynchronous message mode.
	19–20	Reserved.



Note: If the function is completed successfully, an application window receive a message. The message is a return value of RegisterWindowMessage (PCSHLL). The wParam parameter contains Task ID returned by the function call. The HIWORD of lParam parameter contains Return Code 0, which shows the function was successful, and LOWORD of lParam parameter contains function number 126.

Return Code:

The following codes are defined:

Return Code	Explanation
0	The Read Structured Fields function was successful.
1	A specified host presentation space short session ID was not valid or was not connected.
2	An error was made in specifying parameters.
9	A system error occurred.
11	Resource unavailable (memory unavailable).
35	Request rejected. An outbound transmission from the host was canceled.
36	Request rejected. Lost contact with the host.
37	The function was successful, but the host is inbound disabled.

Notes on Using This Function

1. Return code 35 will be returned when the first **Read Structured Fields** or **Write Structured Fields** is requested after an outbound transmission from the host is canceled. Corrective action is the responsibility of the application.
2. Return code 36 requires that the application disconnect from the emulation program and then reconnect to reestablish communication with the host. Corrective action is the responsibility of the application.
3. Return code 37 will be returned if the host is inbound disabled. The **Read Structured Fields** function was successfully requested.
4. The EHLLAPI allows for a maximum of 20 asynchronous requests per application to be outstanding. A return code for unavailable resources (RC=11) is returned if more than 20 asynchronous requests are attempted.

The structured field data contains the application structured fields received from the host. Structured field headers are removed by the EHLLAPI before the structured field data reaches the application.

The structured field data format is as follows:

Offset	Length	Contents
0	1 word	X'0000'.
2	1 word	m (message length: The number of bytes of data in the message, the number does not include the buffer header prefix, which contains 8 bytes). This value is returned by EHLLAPI.
4	1 word	n (buffer size: the supplied length of the data buffer that does include the 8-byte message header). This value must be set by the application.
6	1 word	X'C000'.
8	8 bytes	Length of the first (or only) structured field message.
10	1 byte	First nonlength byte of the structured field message.
		:
m+7	1 byte	Last byte in the structured field message.

Bytes 0 through 7 are the buffer header. These first 8 bytes are used by the emulation program. The user section of the buffer begins with offset 8. Bytes 8 and 9 contain the number of bytes in the first structured field (a structured field message can contain multiple structured fields), including 2 bytes for bytes 8 and 9. Bytes 8 through $m+7$ are used for the structured field message received from the host (which could contain multiple structured fields).

The using application must furnish the complete buffer with the word at offset 0 set to zero. The buffer length must be in the word at offset 4. The word at offset 6 must be X'C000'. The emulation program will place the data message beginning at offset 8 and place the length of the message in the word at offset 2. The buffer length is not disturbed by EHLLAPI.

Synchronous Requests

When **Read Structured Fields** is requested synchronously (the S option in the data string), control is returned to the application only after the request is satisfied. The application can assume:

- The return code is correct.
 - The data in the communications buffer (read buffer) is correct.
 - The host is no longer processing the **Read Structured Fields** request.
-

Asynchronous Requests

When **Read Structured Fields** is requested asynchronously (the A option in the data string), the application *cannot* assume:

- The return code is correct.
- The data in the communications buffer (read buffer) is correct.
- The host is no longer processing the **Read Structured Fields** request.

When requested asynchronously, EHLLAPI returns the following values:

- A 16-bit Request ID in positions 13–14 (9–10 for standard interface) of the data string
- The address of a event object in positions 17–20 of the data string

These are used to complete the asynchronous **Read Structured Fields** call.

The following steps must be completed to determine the outcome of an asynchronous **Read Structured Fields** function call:

- If the EHLLAPI return code is not zero, the request failed. No asynchronous request has been made. The application must take appropriate actions before attempting the call again.
- If the return code is zero, the application should wait until the event object is in the signaled state by using the **Get Request Completion** (125) function or **Wait For Single Object**. The event object should not be reused. The event object is valid only for the duration of the **Read Structured Fields** function call through the completion of the **Get Request Completion** (125) function call.
- Once the event object is in the signaled state, use the returned 16-bit Request ID as the Request ID parameter in a call to the **Get Request Completion** (125) function. The data string returned from the **Get Request Completion** (125) function call contains the final return code of the **Read Structured Fields** function call.

When **Read Structured Fields** is requested asynchronously (the M option in the data string), the application *cannot* assume:

- The return code is correct.
- The data in the communications buffer (read buffer) is correct.
- The host is no longer processing the **Read Structured Fields** request.

When requested asynchronously with the M option, EHLLAPI returns the following values:

- A 16-bit Request ID in positions 13–14 (9–10 for standard interface) of the data string
- Task ID of asynchronous message mode in positions 17–18 (11–12 for standard interface) of the data string.

These are used to complete the asynchronous **Read Structured Fields** call.

Receive File (91)

3270	5250	VT
Yes	Yes	No

The **Receive File** function is used to transfer a file from the host session to the workstation session. It is used the same way as the RECEIVE command is used in the PC/3270. The **Receive File** function can be called by an EHLLAPI application program.

Prerequisite Calls

There are no prerequisite calls for this function.

Call Parameters

	Standard Interface	Enhanced Interface
Function Number	Must be 91.	
Data String	Refer to the examples.	
Length	Length, in number of bytes, of the data string. Overridden if in EOT mode.	

Following are examples of the data strings for a single-byte character set (SBSC):

3270 Session

- To receive the file from the VM/CMS host system:

```
pc_filename [id:] fn ft [fm] [(option)]
```

- To receive the file from the MVS™/TSO host system:

```
pc_filename[id:] dataset[(member)] [/password] [option]
```

- To receive the file from the CICS® host system:

```
pc_filename [id:] host_filename [(option)]
```

5250 Session

- To receive the file from the iSeries™, eServer™ i5, or System i5™ host system:

```
pc_filename [id:] library file member [option]
```

Return Parameters

Return Code	Explanation
2	Parameter error or you have specified a length that is too long (more than 255 bytes) for the EHLLAPI buffer. The file transfer was unsuccessful.
3	File transfer complete.
4	File transfer complete with segmented records.
9	A system error was encountered.
27	File transfer terminated because of either a Cancel button or the timeout set by the Set Session Parameter (9) function.
101	File transfer was successful (transfer to/from CICS®).

If you receive return code 2 or 9, there is a problem with the system or with the way you specified your data string.

Other return codes can also be received, which relate to message numbers generated by the host transfer program. For transfers to a CICS® host transfer program, subtract 100 from the return code to give you the numeric portion of the message. For example, a return code of 101 would mean that the message number INW0001 was issued by the host. For other host transfer programs, just use the return code as the numerical part of the message. For example, a return of 34 would mean that message TRANS34 was issued by the host transfer program. The documentation for your host transfer program should give more information about the meanings of the specific messages.

Operating system error codes reported by EHLLAPI are greater than 300. To determine the error code, subtract 300 and refer to the operating system documentation for return codes.

Notes on Using This Function

1. Four sets of parameters under the **Set Session Parameters** (9) function are related to this function. They are the STRLEN/STREOT, EOT=c, QUIET/NOQUIET and the TIMEOUT=c/TIMEOUT=0 session options. See [items 1 on page 141](#) and [2 on page 141](#) and [items 7 on page 142](#) and [8 on page 143](#) for more information.
2. If no path is specified when the **Receive File** function is executed, the received file is stored in the current subdirectory, which is the directory in which your application is running.

Release (12)

3270	5250	VT
Yes	Yes	Yes

The **Release** function unlocks the keyboard that is associated with the host presentation space reserved using the **Reserve** (11) function.

Prerequisite Calls

Connect Presentation Space (1)

Call Parameters

	Standard Interface	Enhanced Interface
Function Number	Must be 12	
Data String	NA	
Length	NA	
PS Position	NA	

Return Parameters

Return Code	Explanation
0	The Release function was successful.
1	Your program is not connected to a host session.
9	A system error was encountered.

Notes on Using This Function

If you do not **Release** a host presentation space reserved by using the **Reserve** (11) function, you are locked out of that session until you call the **Reset System** (21) function, you call the **Disconnect Presentation Space** (2) function, or you terminate the EHLLAPI application program.

Reserve (11)

3270	5250	VT
Yes	Yes	Yes

The **Reserve** function locks the keyboard that is associated with the host-connected presentation space to block input from the terminal operator.

The reserved host presentation space remains locked until one of the following occurs:

- **Connect** (1) function is executed to a new session.
- **Disconnect Presentation Space** (2) function is executed.
- **Release** (12) function is executed.
- **Reset System** (21) function is executed.
- **Start Keystroke Intercept** (50) function is executed.
- EHLLAPI application program is terminated.

Prerequisite Calls

Connect Presentation Space (1)

Call Parameters

	Standard Interface	Enhanced Interface
Function Number	Must be 11	
Data String	NA	
Length	NA	
PS Position	NA	

Return Parameters

Return Code	Explanation
0	The Reserve function was successful.
1	Your program is not connected to a host session.
5	Presentation space cannot be used.
9	A system error was encountered.

Notes on Using This Function

1. If your EHLLAPI application program is sending a series of transactions to the host, you might need to prevent the user from gaining access to that session until your application processing is complete.
2. The keyboard input that a user makes while the keyboard is locked by this function is enqueued and processed after the session is terminated.
3. This function locks both the mouse and the keyboard input. The application program must unlock the presentation space to enable either the mouse or the keyboard input.

Reset System (21)

3270	5250	VT
Yes	Yes	Yes

The **Reset System** function reinitializes EHLLAPI to its starting state. The session parameter options are reset to their defaults. Event notification is stopped. The reserved host session is released. The host presentation space is disconnected. Keystroke intercept is disabled.

You can use the **Reset System** function during initialization or at program termination to reset the system to a known initial condition.

Prerequisite Calls

There are no prerequisite calls for this function.

Call Parameters

	Standard Interface	Enhanced Interface
Function Number	Must be 21	
Data String	NA	
Length	NA	
PS Position	NA	

Return Parameters

Return Code	Definition
0	The Reset System function was successful.
1	EHLLAPI is not loaded.
9	A system error was encountered.

Notes on Using this Function

For the PC/3270, this function can be used to check whether EHLLAPI is loaded. Place a call to this function at the start of your application and check for a return code of 1.

Search Field (30)

3270	5250	VT
Yes	Yes	Yes

The **Search Field** function examines a field within the connected host presentation space for the occurrence of a specified string. If the target string is found, this function returns the decimal position of the string numbered from the beginning of the host presentation space. (For example, in a 24-row by 80-column presentation space, the row 1, column 1 position is numbered 1 and the row 5, column 1 position is numbered 321.)

This function can be used to search either protected or unprotected fields, but only in a *field-formatted* host presentation space.



Note: If the field at the end of the host presentation space wraps, wrapping occurs when the end of the presentation space is reached.

Prerequisite Calls

Connect Presentation Space (1)

Call Parameters

	Standard Interface	Enhanced Interface
Function Number	Must be 30.	
Data String	Target string for search.	
Length	Length of the target data string. Overridden in EOT mode.	
PS Position	Identifies the target field. For <code>SRCHALL</code> , this can be the PS position of any byte within the target field. For <code>SRCHFROM</code> , it is the beginning point of the search for <code>SRCHFRWD</code> or the ending point of the search for <code>SRCHBKWD</code> . See note 3 on page 126 .	

Return Parameters

This function returns a length and a return code.

Length:

The following codes are defined:

Length	Explanation
= 0	The string was not found.
> 0	The string was found at the indicated host presentation space position.

Return Code:

The following codes are defined:

Return Code	Explanation
0	The Search Field function was successful.
1	Your program is not connected to a host session.
2	Parameter error. Either the string length was zero, or EOT mode was specified but no EOT character was found in calling data string.
7	The host presentation space position is not valid.
9	A system error was encountered.
24	The search string was not found, or the host presentation space was unformatted.

Notes on Using This Function

1. Four sets of parameters under the **Set Session Parameters** (9) function are related to this function. They are the SRCHALL/SRCHFROM, STRLEN/STREOT, SRCHFRWD/SRCHBKWD, and the EOT=c session options. See items 1 on page 141 through 4 on page 142 for more information.
2. You can use the **Set Session Parameters** (9) function to determine whether your searches proceed forward (SRCHFRWD) or backward (SRCHBKWD) in a field.
3. The **Search Field** function normally checks the entire field (SRCHALL default mode). However, you can use the function 9 to specify SRCHFROM. In this mode, the calling PS position parameter does more than identify the target field. It also provides a beginning or ending point for the search.
 - If the SRCHFRWD option is in effect, the search for the designated string begins at the specified PS position and proceeds toward the end of the field.
 - If the SRCHBKWD option is in effect, the search for the designated string begins at the end of the field and proceeds backward toward the specified PS position. If the target string is not found, the search ends at the PS position specified in the calling PS position parameter.



Note: 5250 emulation supports a presentation space of 24 rows by 80 columns. In some instances, Communication Manager 5250 emulation displays a 25th row. This occurs when either an error message from the host is displayed or when the operator selects the SysReq key. Z and I Emulator for Windows displays 25th row information on row 24, or on the status bar. For information to be displayed on the status bar, the status bar must be configured. Refer to *Quick Beginnings* for information on configuring the status bar. By the **EXTEND_PS** option, an EHLLAPI application can use the same interface with Communication Manager EHLLAPI and valid presentation space is extended when this condition occurs.

Search Presentation Space (6)

3270	5250	VT
Yes	Yes	Yes

The **Search Presentation Space** function lets your EHLLAPI program examine the host presentation space for the occurrence of a specified string.

Prerequisite Calls

Connect Presentation Space (1)

Call Parameters

	Standard Interface	Enhanced Interface
Function Number	Must be 6.	
Data String	Target string for search.	

	Standard Interface	Enhanced Interface
Length	Length of the target data string. Overridden in EOT mode.	
PS Position	Position within the host presentation space where the search is to begin (SRCHFRWD option) or to end (SRCHBKWD option). Overridden in SRCHALL (default) mode.	

Return Parameters

This function returns a length and a return code.

Length:

The following codes are defined:

Length	Explanation
= 0	The string was not found.
> 0	The string was found at the indicated host presentation space position.

Return Code:

The following codes are defined:

Return Code	Explanation
0	The Search Presentation Space function was successful.
1	Your program is not connected to a host session.
2	An error was made in specifying parameters.
7	The host presentation space position is not valid.
9	A system error was encountered.
24	The search string was not found.

Notes on Using This Function

- Four sets of parameters under the **Set Session Parameters** (9) function are related to this function. They are the SRCHALL/SRCHFROM, STRLEN/STREOT, SRCHFRWD/SRCHBKWD, and the EOT=c session options. See [items 1 on page 141](#) through [4 on page 142](#) through for more information.
- You can use the **Set Session Parameters** (9) function to specify SRCHBKWD. When this option is in effect, the search operation locates the *last* occurrence of the string.
- The **Search Presentation Space** function normally checks the entire host presentation space. However, you can use the **Set Session Parameters** (9) function to specify SRCHFROM. In this mode, the calling PS position parameter specifies a beginning or ending point for the search.

- If the SRCHFRWD option is in effect, the search for the designated string begins at the specified PS position and proceeds toward the end of the host presentation space.
 - If the SRCHBKWD option is in effect, the search for the designated string begins at the end of the PS and proceeds backward toward the specified PS position. If the target string is not found, the search ends at the PS position specified in the calling PS position parameter.
4. The SRCHFROM option is also useful if you are looking for a keyword that might occur more than once in the host presentation space.
 5. The **Search Presentation Space** function is useful in determining when the host presentation space is available. If your EHLLAPI application is expecting a specific prompt or message before sending data, the **Search Presentation Space** function allows you to check for a prompt message before continuing.



Note: 5250 emulation supports a presentation space of 24 rows by 80 columns. In some instances, Communication Manager 5250 emulation displays a 25th row. This occurs when either an error message from the host is displayed or when the operator selects the SysReq key. Z and I Emulator for Windows displays 25th row information on row 24, or on the status bar. For information to be displayed on the status bar, the status bar must be configured. Refer to *Quick Beginnings* for information on configuring the status bar. By the **EXTEND_PS** option, an EHLLAPI application can use the same interface with Communication Manager EHLLAPI and valid presentation space is extended when this condition occurs.

Send File (90)

3270	5250	VT
Yes	Yes	No

The **Send File** function is used to transfer a file from the workstation session where EHLLAPI is running to a host session.

Prerequisite Calls

There are no prerequisite calls for this function.

Call Parameters

	Standard Interface	Enhanced Interface
Function Number	Must be 90.	
Data String	Refer to the examples.	
Length	Length of the target data string. Overridden in EOT mode.	
PS Position	Must be 0.	

Following are examples of the data strings for SBCS

3270 Session

- To send the file to the VM/CMS host system:

```
pc_filename [id:] fn ft [fm] [(option)]
```

- To send the file to the MVS/TSO host system:

```
pc_filename [id:] dataset[(member)] [/password] [option]
```

- To send the file to the CICS host system:

```
pc_filename [id:] host_filename [(option)]
```

5250 Session

- To send the file to the iSeries™, eServer™ i5, or System i5™ host system:

```
pc_filename [id:] library file member [option]
```

Return Parameters

Return Code	Explanation
2	Parameter error or you have specified a length that is too long (more than 255 bytes) for the EHLLAPI buffer. The file transfer was unsuccessful.
3	File transfer complete.
4	File transfer complete with segmented records.
5	Workstation file name is not valid or not found. File transfer was canceled.
9	A system error was encountered.
27	File transfer terminated because of either a Cancel button or the timeout set by the Set Session Parameter (9) function.
101	File transfer was successful (transfer to/from CICS).

If you receive return code 2 or 9, there is a problem with the system or with the way you specified your data string.

Other return codes can also be received which relate to message numbers generated by the host transfer program. For transfers to a CICS host transfer program, subtract 100 from the return code to give you the numeric portion of the message. For example, a return code of 101 would mean that the message number INW0001 was issued by the host. For other host transfer programs, just use the return code as the numerical part of the message. For example, a return of 34 would mean that message TRANS34 was issued by the host transfer program. The documentation for your host transfer program should give more information about the meanings of the specific messages.

Operating system error codes reported by EHLLAPI are greater than 300. To determine the error code, subtract 300 and refer to the operating system documentation for return codes.

Notes on Using This Function

1. Four sets of parameters under the **Set Session Parameters** (9) function are related to this function. They are the QUIET/NOQUIET, STRLEN/STREOT, TIMEOUT=c/TIMEOUT=0, and the EOT=c session options. See [items 1 on page 141](#) and [2 on page 141](#) plus [items 7 on page 142](#) and [8 on page 143](#) for more information.

Send Key (3)

3270	5250	VT
Yes	Yes	Yes

The **Send Key** function is used to send either a keystroke or a string of keystrokes to the host presentation space.

You define the string of keystrokes to be sent with the calling data string parameter. The keystrokes appear to the target session as though they were entered by the terminal operator. You can also send all attention identifier (AID) keys such as Enter and so on. All host fields that are input protected or are numeric only must be treated accordingly.

Prerequisite Calls

Connect Presentation Space (1)

Call Parameters

	Standard Interface	Enhanced Interface
Function Number	Must be 3.	
Data String	A string of keystrokes, maximum 255. Uppercase and lowercase ASCII characters are represented literally. Function keys and shifted function keys are represented by mnemonics. See Keyboard Mnemonics on page 132 .	
Length	Length of the source data string. Overridden if in EOT mode.	
PS Position	NA	

Return Parameters

Return Code	Explanation
0	The keystrokes were sent; status is normal.
1	Your program is not connected to a host session.
2	An incorrect parameter was passed to EHLLAPI.
4	The host session was busy; all of the keystrokes could not be sent.
5	Input to the target session was inhibited or rejected; all of the keystrokes could not be sent.
9	A system error was encountered.

Notes on Using This Function

1. The parameters under the **Set Session Parameters** (9) function are related to this function. They are the **AUTORESET/NORESET**, **STRLIN/STREOT**, **EOT=c**, **ESC=c**, and **RETRY/NORETRY** session options. See items [1 on page 141](#) and [2 on page 141](#), [9 on page 144](#) and [10 on page 144](#), and [19 on page 147](#) for more information.
2. Keystrokes cannot be sent to the host session when the keyboard is locked or busy. You can check this condition with the **Wait** (4) function.
3. If the host is busy, input might be rejected.
4. The length of the data string must be explicitly defined by the default length parameter, but it can be defined implicitly by the **EOT=c** option of the **Set Session Parameters** (9) function.

When explicitly defining length (see item 1), the value for the length parameter passed by the application must be calculated. For this calculation, allow 2 bytes for compound keystrokes such as `@E` and allow 4 bytes for compound keystrokes such as `@A@C`.

5. To send special control keys, a compound character coding scheme is used. In this coding scheme, one keystroke is represented by a sequence of two to four ASCII characters. The first and third character are always the escape character. The second and fourth character are always a keycode.

To send the sequence `LOGON ABCDE` followed by the Enter key, you would code the string `LOGON ABCDE@E`. A complete list of these keycodes is represented in [Keyboard Mnemonics on page 132](#).

This compound coding technique allows an ASCII string representation of all necessary keystroke codes without requiring the use of complex hexadecimal key codes.

The default escape character is `@`. The value of the escape character can be changed to any other character with the **ESC=c** option of the **Set Session Parameters** (9) function.

6. Users needing higher levels of performance should use the **Copy String to Field** (33) or **Copy String to Presentation Space** (15) function rather than send keystrokes with the **Send Key** (3) function. But remember, only the **Send Key** (3) function can send the special control keys.
7. Refer to [Set Session Parameters \(9\) on page 139](#) session option [10 on page 144](#) (**NORESET** option) to improve the performance of this function.

Unless **NORESET** is required, the reset mnemonic is added to the keystroke strings as a prefix. Therefore, all resettable status except input inhibit are reset.

The **NORESET** option is not the same as the **Reset System** (21) function.

8. The keystroke strings, including the AID key, are sent to the host via multiple paths. Each path sends the strings before the first AID key (or including the AID key). EHLLAPI adjusts the string length and the start position of each path. For a host application program, any keystroke might be lost by the AID key process. Therefore, you should not send a keystroke list that includes plural AID keys.

9. During the @P (Print) or @A@T (Print Presentation Space) process, all requests that update the presentation space are rejected. If the presentation space is busy or the interruption request occurs during the print request, the mnemonic @A@R (Device Reset – Cancel to print the Presentation Space) cancels the request and resets the status.

Keyboard Mnemonics

The keyboard mnemonics provide the ASCII characters representing the special function keys of the keyboard in the workstation. The abbreviation codes make the mnemonics for special keys easy to remember. An alphabetic key code is used for the most common keys. For example, the **Clear** key is **C**, and the **Tab** key is **T**.

Table 7: Mnemonics with Uppercase Alphabetic Characters on page 132 shows the mnemonics using uppercase alphabetic characters:

Table 7. Mnemonics with Uppercase Alphabetic Characters

Mnemonic	Meaning	3270	5250	VT
@B	Left Tab	Yes	Yes	No
@C	Clear	Yes	Yes	No
@D	Delete	Yes	Yes	No
@E	Enter	Yes	Yes	No
@F	Erase EOF	Yes	Yes	No
@H	Help	No	Yes	No
@I	Insert	Yes	Yes	No
@J	Jump (Set Focus)	Yes	Yes	No
@L	Cursor Left	Yes	Yes	Yes
@N	New Line	Yes	Yes	Yes
@O	Space	Yes	Yes	Yes
@P	Print	Yes	Yes	Yes
@R	Reset	Yes	Yes	No
@T	Right Tab	Yes	Yes	Yes
@U	Cursor Up	Yes	Yes	Yes
@V	Cursor Down	Yes	Yes	Yes
@Z	Cursor Right	Yes	Yes	Yes

Table 8: Mnemonics with Numbers or Lowercase Characters on page 132 shows the mnemonics using a number or lowercase alphabetic characters.

Table 8. Mnemonics with Numbers or Lowercase Characters

Mnemonic	Meaning	3270	5250	VT
@0	Home	Yes	Yes	No
@1	PF1/F1	Yes	Yes	No

Table 8. Mnemonics with Numbers or Lowercase Characters (continued)

Mnemonic	Meaning	3270	5250	VT
@2	PF2/F2	Yes	Yes	No
@3	PF3/F3	Yes	Yes	No
@4	PF4/F4	Yes	Yes	No
@5	PF5/F5	Yes	Yes	No
@6	PF6/F6	Yes	Yes	Yes
@7	PF7/F7	Yes	Yes	Yes
@8	PF8/F8	Yes	Yes	Yes
@9	PF9/F9	Yes	Yes	Yes
@a	PF10/F10	Yes	Yes	Yes
@b	PF11/F11	Yes	Yes	Yes
@c	PF12/F12	Yes	Yes	Yes
@d	PF13	Yes	Yes	Yes
@e	PF14	Yes	Yes	Yes
@f	PF15	Yes	Yes	Yes
@g	PF16	Yes	Yes	Yes
@h	PF17	Yes	Yes	Yes
@i	PF18	Yes	Yes	Yes
@j	PF19	Yes	Yes	Yes
@k	PF20	Yes	Yes	Yes
@l	PF21	Yes	Yes	No
@m	PF22	Yes	Yes	No
@n	PF23	Yes	Yes	No
@o	PF24	Yes	Yes	No
@q	End	Yes	Yes	No
@u	Page Up	No	Yes	No
@v	Page Down	No	Yes	No
@x	PA1	Yes	Yes	No
@y	PA2	Yes	Yes	No
@z	PA3	Yes	Yes	No

Table 9: Mnemonics with @A and @ Uppercase Alphabetic Characters on page 133 shows the mnemonics using the combination @A and @alphabetic uppercase (A–Z) key.

Table 9. Mnemonics with @A and @ Uppercase Alphabetic Characters

Mnemonic	Meaning	3270	5250	VT
@A@C	Test	No	Yes	No
@A@D	Word Delete	Yes	Yes	No
@A@E	Field Exit	Yes	Yes	No

Table 9. Mnemonics with @A and @ Uppercase Alphabetic Characters (continued)

Mnemonic	Meaning	3270	5250	VT
@A@F	Erase Input	Yes	Yes	No
@A@H	System Request	Yes	Yes	No
@A@I	Insert Toggle	Yes	Yes	No
@A@J	Cursor Select	Yes	Yes	No
@A@L	Cursor Left Fast	Yes	Yes	No
@A@Q	Attention	Yes	Yes	No
@A@R	Device Cancel (Cancels Print Presentation Space)	Yes	Yes	No
@A@T	Print Presentation Space	Yes	Yes	Yes
@A@U	Cursor Up Fast	Yes	Yes	No
@A@V	Cursor Down Fast	Yes	Yes	No
@A@Z	Cursor Right Fast	Yes	Yes	No

Table 10: Mnemonics with @A and @ Lowercase Alphabetic Characters on page 134 shows the mnemonics using the combination @A and @number or @A and @alphabetic lowercase (a–z) key.

Table 10. Mnemonics with @A and @ Lowercase Alphabetic Characters

Mnemonic	Meaning	3270	5250	VT
@A@9	Reverse Video	Yes	Yes	No
@A@b	Underscore	Yes	No	No
@A@c	Reset Reverse Video	Yes	No	No
@A@d	Red	Yes	No	No
@A@e	Pink	Yes	No	No
@A@f	Green	Yes	No	No
@A@g	Yellow	Yes	No	No
@A@h	Blue	Yes	No	No
@A@i	Turquoise	Yes	No	No
@A@j	White	Yes	No	No
@A@l	Reset Host Colors	Yes	No	No
@A@t	Print (Personal Computer)	Yes	Yes	No
@A@y	Forward Word Tab	Yes	Yes	No
@A@z	Backward Word Tab	Yes	Yes	No

Table 11: Mnemonics with @A and @ Alphanumeric (Special) Characters on page 135 shows the mnemonics using the combination @A and @special character.

Table 11. Mnemonics with @A and @ Alphanumeric (Special) Characters

Mnemonic	Meaning	3270	5250	VT
@A@-	Field -	No	Yes	No
@A@+	Field +	No	Yes	No
@A@<	Record Backspace	No	Yes	No

Table 12: Mnemonics with @S (Shift), @W (Edit) and @ Alphanumeric Characters on page 135 shows the mnemonics using the combination @S, @W, and @alphanumeric lowercase.

Table 12. Mnemonics with @S (Shift), @W (Edit) and @ Alphanumeric Characters

Mnemonic	Meaning	3270	5250	VT
@S@E	Print Presentation Space on Host	No	Yes	No
@S@x	Dup	Yes	Yes	No
@S@y	Field Mark	Yes	Yes	No
@W@C	Edit Copy	Yes	Yes	Yes
@W@D	Edit Clear	Yes	Yes	Yes
@W@E	Edit Copy Append	Yes	Yes	Yes
@W@L	Edit Copy Link	Yes	Yes	Yes
@W@N	Edit Paste Next	Yes	Yes	Yes
@W@V	Edit Paste	Yes	Yes	Yes
@W@X	Edit Cut	Yes	Yes	Yes
@W@Z	Edit Undo	Yes	Yes	Yes



Note: @W Edit mnemonics are supported only in EHLLAPI functions in Enhanced mode. See Start Keystroke Intercept function under [Summary of EHLLAPI Functions on page 32](#).

VT Only: Table 13: Mnemonics Using @M, @Q and @Alphanumeric Lowercase (For VT Only) on page 135 shows the mnemonics using the combination @M and @number or @alphanumeric lowercase (a-z)

Table 13. Mnemonics Using @M, @Q and @Alphanumeric Lowercase (For VT Only)

Mnemonic	Meaning	3270	5250	VT
@M@0	VT Numeric Pad 0	No	No	Yes
@M@1	VT Numeric Pad 1	No	No	Yes

Table 13. Mnemonics Using @M, @Q and @Alphabetic Lowercase (For VT Only) (continued)

Mnemonic	Meaning	3270	5250	VT
@M@2	VT Numeric Pad 2	No	No	Yes
@M@3	VT Numeric Pad 3	No	No	Yes
@M@4	VT Numeric Pad 4	No	No	Yes
@M@5	VT Numeric Pad 5	No	No	Yes
@M@6	VT Numeric Pad 6	No	No	Yes
@M@7	VT Numeric Pad 7	No	No	Yes
@M@8	VT Numeric Pad 8	No	No	Yes
@M@9	VT Numeric Pad 9	No	No	Yes
@M@-	VT Numeric Pad -	No	No	Yes
@M@,	VT Numeric Pad ,	No	No	Yes
@M@.	VT Numeric Pad .	No	No	Yes
@M@e	VT Numeric Pad Enter	No	No	Yes
@M@f	VT Edit Find	No	No	Yes
@M@i	VT Edit Insert	No	No	Yes
@M@r	VT Edit Remove	No	No	Yes
@M@s	VT Edit Select	No	No	Yes
@M@p	VT Edit Previous Screen	No	No	Yes
@M@n	VT Edit Next Screen	No	No	Yes
@M@a	VT PF1	No	No	Yes
@M@b	VT PF2	No	No	Yes
@M@c	VT PF3	No	No	Yes
@M@d	VT PF4	No	No	Yes
@M@h	VT HOld Screen	No	No	Yes
@M@(space)	Control Code NUL	No	No	Yes
@M@A	Control Code SOH	No	No	Yes
@M@B	Control Code STX	No	No	Yes
@M@C	Control Code ETX	No	No	Yes
@M@D	Control Code EOT	No	No	Yes
@M@E	Control Code ENQ	No	No	Yes
@M@F	Control Code ACK	No	No	Yes
@M@G	Control Code BEL	No	No	Yes
@M@H	Control Code BS	No	No	Yes
@M@I	Control Code HT	No	No	Yes
@M@J	Control Code LF	No	No	Yes
@M@K	Control Code VT	No	No	Yes
@M@L	Control Code FF	No	No	Yes

Table 13. Mnemonics Using @M, @Q and @Alphabetic Lowercase (For VT Only) (continued)

Mnemonic	Meaning	3270	5250	VT
@M@M	Control Code CR	No	No	Yes
@M@N	Control Code SO	No	No	Yes
@M@O	Control Code SI	No	No	Yes
@M@P	Control Code DLE	No	No	Yes
@M@Q	Control Code DC1	No	No	Yes
@M@R	Control Code DC2	No	No	Yes
@M@S	Control Code DC3	No	No	Yes
@M@T	Control Code DC4	No	No	Yes
@M@U	Control Code NAK	No	No	Yes
@M@V	Control Code SYN	No	No	Yes
@M@W	Control Code ETB	No	No	Yes
@M@X	Control Code CAN	No	No	Yes
@M@Y	Control Code EM	No	No	Yes
@M@Z	Control Code SUB	No	No	Yes
@M@u	Control Code ESC	No	No	Yes
@M@v	Control Code FS	No	No	Yes
@M@w	Control Code GS	No	No	Yes
@M@x	Control Code RS	No	No	Yes
@M@y	Control Code US	No	No	Yes
@M@z	Control Code DEL	No	No	Yes
@Q@A	VT User Defined Key 6	No	No	Yes
@Q@B	VT User Defined Key 7	No	No	Yes
@Q@C	VT User Defined Key 8	No	No	Yes
@Q@D	VT User Defined Key 9	No	No	Yes
@Q@E	VT User Defined Key 10	No	No	Yes
@Q@F	VT User Defined Key 11	No	No	Yes
@Q@G	VT User Defined Key 12	No	No	Yes
@Q@H	VT User Defined Key 13	No	No	Yes
@Q@I	VT User Defined Key 14	No	No	Yes
@Q@J	VT User Defined Key 15	No	No	Yes
@Q@K	VT User Defined Key 16	No	No	Yes

Table 13. Mnemonics Using @M, @Q and @Alphabetic Lowercase (For VT Only) (continued)

Mnemonic	Meaning	3270	5250	VT
@Q@L	VT User Defined Key 17	No	No	Yes
@Q@M	VT User Defined Key 18	No	No	Yes
@Q@N	VT User Defined Key 19	No	No	Yes
@Q@O	VT User Defined Key 20	No	No	Yes
@Q@a	VT Backtab	No	No	Yes
@Q@r	VT Clear Page	No	No	Yes
@Q@s	VT Edit	No	No	Yes

The following table shows the mnemonics using a special character.

Table 14. Mnemonics with Special Character Keys

Mnemonic	Meaning	3270	5250	VT
@@	@	Yes	Yes	Yes
@\$	Alternate Cursor (The Presentation Manag- er® Interface only)	Yes	Yes	Yes
@<	Backspace	Yes	Yes	Yes

The following character keys are interpreted as they are.

a-z	!	'	'	<	}
A-Z	\$	(.	>	[
0-9	%)	/	=]
~	&	*	:	?	
#	"	+	;	{	

Set Cursor (40)

3270	5250	VT
Yes	Yes	Yes

The **Set Cursor** function is used to set the position of the cursor within the host presentation space. Before using the **Set Cursor** function, a workstation application must be connected to the host presentation space.

Prerequisite Calls

Connect Presentation Space (1)

Call Parameters

	Standard Interface	Enhanced Interface
Function Number	Must be 40	
Data String	NA	
Length	NA	
PS Position	Desired cursor position in the connected host presentation space	

Return Parameters

Return Code	Explanation
0	Cursor was successfully located at the specified position.
1	Your program is not connected to a host session.
4	The session is busy.
7	A cursor location less than 1 or greater than the size of the connected host presentation space was specified.
9	A system error occurred.

Set Session Parameters (9)

3270	5250	VT
Yes	Yes	Yes

The **Set Session Parameters** function lets you change certain default session options in EHLLAPI for all sessions. When EHLLAPI is loaded, the default settings for session options are as indicated by the underscored entries in the tables that appear in [Session Options on page 141](#) . Any, some, or all of these settings can be changed by including the desired option in the calling data string as explained below. Specified settings remain in effect until:

- Changed by a subsequent **Set Session Parameters** (9) function that specifies a new value.
- The **Reset System** (21) function is executed.
- The EHLLAPI application program is terminated.

The following table lists those EHLLAPI functions that are affected by session options. Functions not listed in the table are not affected by any of the session options. Session options that affect each function are indicated by corresponding entries in the “*See Items*” column. These entries are indexed to the list that follows [Call Parameters on page 141](#).

Function Number	Function Name	See Items
1	Connect Presentation Space	11 on page 144 , 21 on page 147 , 22 on page 148

Function Number	Function Name	See Items
3	Send Key	1 on page 141, 2 on page 141, 9 on page 144, 10 on page 144, 19 on page 147
4	Wait	12 on page 144
5	Copy Presentation Space	5 on page 142, 13 on page 145, 14 on page 146, 15 on page 146, 17 on page 146, 20 on page 147
6	Search Presentation Space	1 on page 141, 2 on page 141, 3 on page 142, 4 on page 142
8	Copy Presentation Space to String	5 on page 142, 13 on page 145, 14 on page 146, 15 on page 146, 17 on page 146, 20 on page 147
10	Query Sessions	16 on page 146, 20 on page 147
15	Copy String to Presentation Space	1 on page 141, 2 on page 141, 13 on page 145, 14 on page 146, 18 on page 146, 20 on page 147
18	Pause	6 on page 142
30	Search Field	1 on page 141, 2 on page 141, 3 on page 142, 4 on page 142, 20 on page 147
33	Copy String to Field	1 on page 141, 2 on page 141, 13 on page 145, 14 on page 146, 18 on page 146, 20 on page 147
34	Copy Field to String	5 on page 142, 13 on page 145, 14 on page 146, 17 on page 146, 20 on page 147
35	Copy Presentation Space to Clipboard	5 on page 142, 13 on page 145, 14 on page 146, 17 on page 146, 20 on page 147
36	Paste Clipboard to Presentation Space	1 on page 141, 2 on page 141, 13 on page 145, 14 on page 146, 18 on page 146, 20 on page 147
51	Get Key	9 on page 144, 12 on page 144
90	Send File	1 on page 141, 2 on page 141, 7 on page 142, 8 on page 143
91	Receive File	1 on page 141, 2 on page 141, 7 on page 142, 8 on page 143
101	Connect Window Services	21 on page 147, 22 on page 148

Prerequisite Calls

There are no prerequisite calls for this function.

Call Parameters

	Standard Interface	Enhanced Interface
Function Number	Must be 9.	
Data String	String containing the desired values of those session options that are to be changed. The data string can contain any of the values in the tables of Session Options on page 141 . The values should be placed on the data string line, separated by commas or blanks. The sets of parameters are explained in terms of the functions they affect.	
Length	Explicit length of the source data string (the <code>STREOT</code> option is not allowed).	
PS Position	NA.	

Session Options

The following tables show the session options. The default is underlined.

- The values in the following table determine how the data string length is defined for functions **Send Key** (3), **Search Presentation Space** (6), **Copy String to Presentation Space** (15), **Search Field** (30), **Copy String to Field** (33), **Send File** (90), and **Receive File** (91).

Value	Explanation
<u>STRLEN</u>	An explicit length is passed for all strings.
STREOT	Lengths are not explicitly coded. Calling (source) data strings are terminated with an EOT character.

- The statement in the following table is used to specify the character that is used as the end-of-text (EOT) delimiter in the calling (source) data string for EHLLAPI functions **Send Key** (3), **Search Presentation Space** (6), **Copy String to Presentation Space** (15), **Search Field** (30), **Copy String to Field** (33), **Send File** (90), and **Receive File** (91).

3. The values in the following table affect the **Search Presentation Space** (6) and **Search Field** (30) search functions.

4. The values in the following table affect the **Search Presentation Space** (6) and **Search Field** (30) search functions. They determine the direction for the search.

5. The values in the following table determine how attribute bytes are treated for functions **Copy Presentation Space** (5), **Copy Presentation Space to String** (8), and **Copy Field to String** (34).

6. The values in the following table affect the **Pause** (18) function.

	After the Start Host Notification (23) function is executed, a host event satisfies a pause.

7. The values in the following table determine whether messages generated by file transfer functions **Send File** (90) and **Receive File** (91) are displayed.

8. The statements in the following table determine how long Z and I Emulator for Windows EHLLAPI waits before it automatically issues a Cancel during execution of file transfer functions **Send File** (90) and **Receive File** (91). To be valid, **c** must be a capital letter J–N and must not be preceded by a blank.

Value	Explanation
	L 6.0 M 6.5 N 7.0

9. The statement in the following table is used to define the escape character for keystroke mnemonics. This session option affects functions **Send Key** (3) and **Get Key** (51). The value of `␣` must be entered as a 1-byte literal character string with no preceding blanks.

10. The values in the following table determine whether EHLLAPI automatically precedes strings sent using the **Send Key** (3) function with a reset.

11. The values in the following table affect the manner in which the **Connect Presentation Space** (1) command function.

Value	Explanation
CONLOG	Establishes a logical connection between the workstation session and a host session. During Connect, does not jump to the requested presentation space.
CONPHYS	Establishes a physical connection between the workstation session and a host session. During Connect, jumps to the requested presentation space.

12. The values in the following table affect the **Wait** (4) function and **Get Key** (51) function. For each value, there are two different effects, one for each function.

Value	Explanation
	For the Get Key (51) function, does not return control to your EHLLAPI application program until it has intercepted a key (normal or AID key based on the option specified under the Start Keystroke Intercept (50) function).
LWAIT	<p>For the Wait (4) function, waits until XCLOCK (X [])/XSYSTEM clears. This option is not recommended, because control does not return to your application until the host is available.</p> <p>For the Get Key (51) function, does not return control to your EHLLAPI application program until it has intercepted a key (normal or AID key based on the option specified under the Start Keystroke Intercept (50) function).</p>
NWAIT	<p>For the Wait (4) function, checks status and returns immediately (no wait).</p> <p>For the Get Key (51) function, returns return code 25 (keystrokes not available) in the fourth parameter if nothing is queued matching the option specified under the Start Keystroke Intercept (50) function.</p>



Note: Use of NWAIT is recommended.

13. The values in the following table affect **Copy Presentation Space** (5), **Copy Presentation Space to String** (8), **Copy String to Presentation Space** (15), **Copy String to Field** (33), and **Copy Field to String** (34). Extended attribute bytes (EAB) include extended character attributes and extended field attributes.



text is displayed, but not visible to the human operator. The only way for your application to detect this is to use the EAB and XLATE session parameters and then copying the PS. The foreground/background color of each position is returned and you can determine which characters are invisible.

14. The values in the following table affect **Copy Presentation Space** (5), **Copy Presentation Space to String** (8), **Copy String to Presentation Space** (15), **Copy String to Field** (33), and **Copy Field to String** (34).

15. The values in the following table affect **Copy Presentation Space** (5), **Copy Presentation Space to String** (8) and **Copy Presentation Space to Clipboard** (35) if NOATTRB and NOEAB are specified.

16. The values in the following table affect the presentation space size that is returned by the **Query Sessions** (10).

Value	Explanation
CFGSIZE	Returns the configured size of the connected presentation space. This option ignores any override of the configured size by the host.
NOCFGSIZE	Returns the current size of the connected presentation space.

17. The values in the following table affect **Copy Presentation Space** (5), **Copy Presentation Space to String** (8), **Copy Field to String** (34) and **Copy Presentation Space to Clipboard** (35).

Value	Explanation
DISPLAY	Copy nondisplay fields in the presentation space to the target buffer area in the same manner as display fields. Current applications function normally.
NODISPLAY	Do not copy nondisplay fields in the presentation space to the target buffer area. Copy the nondisplay fields to the target buffer as a string of null characters. This allows applications to display the copied buffers in the presentation widow without displaying confidential information, such as passwords.

18. The values in the following table affect **Copy String to Presentation Space** (15), **Copy String to Field** (33) and **Paste Clipboard to Presentation Space** (36).

Value	Explanation
<u>NOPUTEAB</u>	EAB is not contained in the data string of Copy String to Presentation Space or Copy String to Field .
PUTEAB	EAB is contained with character data in the data string of Copy String to Presentation Space or Copy String to Field .

This option is used for the compatibility with Communication Manager/2. For Communication Manager/2, the data string, which is specified in **Copy String to Presentation Space** or **Copy String to Field**, must contain EAB (or EAD) with character data when EAB (or EAD) is valid in **Set Session Parameters**. Whereas, for the previous Z and I Emulator for Windows, the data string specified in these functions must consist of character data only even if EAB (or EAD) is valid. But Z and I Emulator for Windows allows that the data string contains EAB (or EAD) by setting PUTEAB to provide the compatibility with Communication Manager/2.

19. The values in the following table affect the **Send Key** (3) function. Keystrokes are not processed if the keyboard is blocked or in use. The options determine whether the function tries to resend the keystrokes until a 4-minute timeout occurs or if the function returns immediately after determining the keyboard is blocked or in use.

Value	Explanation
<u>RETRY</u>	Continues to attempt to send keystrokes until they are sent or until a 4-minute timeout occurs.
NORETRY	Returns immediately after determining the keyboard is blocked or in use.

20. The values in the following table affect **Copy Presentation Space** (5), **Copy Presentation Space to String** (8), **Copy String to Presentation Space** (15), **Copy String to Field** (33), **Copy Field to String** (34) **Search Field** (30), **Query Sessions** (10), **Copy Presentation Space to Clipboard** (35) and **Paste Clipboard to Presentation Space** (36).

Value	Explanation
EXTEND_PS	5250 emulation supports a presentation space of 24 rows by 80 columns. In some instances, Communication Manager 5250 emulation displays a 25th row. This occurs when either an error message from the host is displayed or when the operator selects the SysReq key. Z and I Emulator for Windows displays 25th row information on row 24, but EHLLAPI normally sees the <i>real</i> 24th row. By EXTEND_PS option, an EHLLAPI application can use the same interface with Communication Manager EHLLAPI and valid presentation space is extended when this condition occurs.
<u>NOEXTEND_PS</u>	The presentation space is not extended when the above condition occurs. This is the default value.

21. The values in the following table affect the **Connect Presentation Space** (1) and **Connect Window Services** (101) functions. The options specify whether an application can or will share the presentation space to which

it is connected with another application. Only one of the following values can be specified with each **Set Session Parameter** call.

Value	Explanation
SUPER_WRITE	The application allows other applications that allow sharing and have write access permissions to concurrently connect to the same presentation space. The originating application performs supervisory-type functions but does not create errors for other applications that share the presentation space.
WRITE_SUPER	The application requires write access and allows only supervisory application to concurrently connect to its presentation space. This is the default value.
WRITE_WRITE	The application requires write access and allows partner or other applications with predictable behavior to share the presentation space.
WRITE_READ	The application requires write access and allows other applications that perform read-only functions to share the presentation space. The application is also allowed to copy the presentation space and perform other read-only operations as usual.
WRITE_NONE	The application has exclusive use of the presentation space. No other applications are allowed to share the presentation space, including supervisory applications. The application is allowed to copy the presentation space and perform read-only operations as usual.
READ_WRITE	The application requires only read access to monitor the presentation space and allows other applications that perform read or write, or both, functions to share the presentation space. The application is also allowed to copy the presentation space and perform other read-only operations as usual.

22. The values in the following table allow applications that have presentation space sharing requirements to limit the sharing to a partner application (an application that was developed to work with it).

Value	Explanation
NOKEY	Allows the application to be compatible with existing applications that do not specify the KEY parameter.
KEY\$nnnnnnnn	Uses a keyword to restrict sharing access to the presentation space that it supports. The keyword must be exactly 8 bytes in length.

Return Parameters

This function returns a length and a return code.

Length:

Number of valid session parameters that are set.

Return Code:

The following codes are defined:

Return Code	Explanation
0	The session parameters have been set.
2	One or more parameters were not valid.
9	A system error was encountered.

Start Close Intercept (41)

3270	5250	VT
Yes	Yes	Yes

The **Start Close Intercept** function allows the application to intercept close requests generated when a user selects the close option from the emulator session window. This function intercepts the close request and discards it until a **Stop Close Intercept** (43) function is requested.

After using this function, your application program can use the **Query Close Intercept** (42) function to determine when a close request has occurred.

Prerequisite Calls

There are no prerequisite calls for this function.

Call Parameters

Byte	Definition	
	Standard Interface	Enhanced Interface
Function Number	Must be 41	
Data String	See the following table	
Length	5 or 6	Must be 12
PS Position	NA	

The data string contains the following items.

Byte		Definition
Standard	Enhanced	
1	1	A 1-character presentation space short name (PSID).
	2–4	Reserved.
4–5		The data in these positions is ignored by EHLLAPI. However, no error is caused if the migrating program has data in these positions. This data is accepted to provide compatibility with migrating applications.
6	5	Specify M to request asynchronous message mode (Windows only).

Byte		Definition
	6–8	Reserved.
2–3	9–12	When M is specified in position 5 (6 for 16-bit), the window handle of the window that receives the message should be set. The message is a return value of RegisterWindowMessage (PCSHLL) (not equal 0).

Return Parameters

This function returns a data string and a return code.

Data String:

If asynchronous message mode is not specified in position 5 (6 for standard interface) and the function is completed successfully, the following data string is returned.

Byte		Definition
Standard	Enhanced	
1	1	A 1-character presentation space short name (PSID).
	2–8	Reserved.
	9–12	4 byte value in which the event object address is returned by EHLLAPI. The application can wait for this event object. (32-bit only).

Data String:

If M (asynchronous message mode) is specified in position 5 (6 for standard interface) and the function is completed successfully, the following data string is returned.

Byte		Definition
Standard	Enhanced	
1	1	A 1-character presentation space short name (PSID)
	2–8	Reserved
2–3	9–10	Task ID of asynchronous message mode



Note: If a user selects the close option, an application window receives a message. The message is a return value of RegisterWindowMessage (PCSHLL). The wParam parameter will contain the Task ID returned by this function call. The HIWORD of the lParam parameter will contain the Return Code 26, which shows a close intercept occurred, and the LOWORD of the lParam parameter will contain the function number 41.

Return Code:

The following codes are defined:

Return Code	Explanation
0	The Start Close Intercept function was successful.
1	An incorrect host presentation space was specified.

Return Code	Explanation
2	A parameter error occurred.
9	A system error occurred.
10	The function is not supported by the emulation program.

Notes on Using This Function

1. The returned event object or semaphore is in a non-signaled state when the start request function returns. The event object is in the signaled state each time a close request occurs. To receive notification of multiple close request events, put the event object into the signaled state each time using **SetEvent** or the **Query Close Intercept (42)** function.
2. After using this function, your application program can use the **Query Close Intercept (42)** function to determine when a close request has occurred. The application can wait on the returned event object to determine when the event has occurred.
3. This is not an exclusive call. Multiple applications can request this function for the same short session ID.
4. If there are no applications intercepting close requests for a session, any subsequent close requests selected by the user from the emulator operations dialog result in a normal stop requested for that session.

Start Communication Notification (80)

3270	5250	VT
Yes	Yes	Yes

The **Start Communication Notification** function begins the process by which your EHLLAPI application can determine whether the specified session is connected to a host.

After using this function, the application can use **Query Communication Event (81)** to determine whether the session is connected or disconnected.

Prerequisite Calls

There are no prerequisite calls for this function.

Call Parameters

	Enhanced Interface
Function Number	Must be 80
Data String	Preallocated structure; see the following table
Length	16
PSPosition	NA

The calling data structure contains these elements

Byte	Definition
1	A 1-character presentation space short name (PSID).
2-4	Reserved
5	One of the following values: <ul style="list-style-type: none"> • The character C asks for notification when the session either disconnects or connects to the host. • The character A requests the asynchronous mode of notification. When A is specified, position 9-12 returns the address of an event object (Windows). The character C must be placed in position 13. • The character M requests the asynchronous message mode of the notification. When M is specified, the event selection character C must be placed in position 13.
6-8	Reserved
9-12	When M is specified in position 5, the window handle of the window that receives the message should be set. The message is a return value of RegisterWindowMessage (PCSHLL)—(not zero).
13	This should contain the character C if position 5 is A or M.
14-16	Reserved

Data String

If A (asynchronous mode) is specified in position 5 of the calling data structure and the function is completed successfully, the following data string is returned:

Byte	Definition
1	A 1-character presentation space short-name (PSID)
2-8	Reserved
9-12	4-byte binary value in which the event object handle is returned by EHLLAPI. The application can wait for this event object.

If M (asynchronous message mode) is specified in position 5 of the calling data structure and the function is completed successfully, the following data string is returned:

Byte	Definition
1	A 1-character presentation space short-name (PSID)
2-8	Reserved
9-10	Task ID of asynchronous message mode

When the session connects or disconnects an application window receives a message. The message is the return value of RegisterWindow Message (PCSHLL). The wParam contains the Task ID returned by the function call.

HIWORD of IPParam contains a 21 if the session is connected to the host or a 22 if the session is disconnected. The LOWORD of IPParam contains the function number 80.

Return Parameters

Return Code	Definition
0	The function was successful
1	An incorrect PSID was specified
2	An error was made in designating parameters
9	A system error was encountered

Notes on using this Function

1. An application program can issue this function for multiple host sessions. The **Query Communication Event** (81) function can be used to determine the session communication status.
2. If the application chooses the asynchronous option, it can use the Windows SDK call **WaitForSingleObject** to wait until the sessions communication status has changed.
3. The event object is initially in a non-signaled state. It is signaled each time an event occurs. To receive notification for multiple events the application must put the event object into the non-signaled state each time it is signaled, by using the Windows SDK call **ResetEvent**, or by using function 81 **Query Communications Event**.
4. Multiple calls to this function with the same options from the same application will be ignored.
5. This is not exclusive to one application. Several applications can request this function for the same Session ID.

Start Host Notification (23)

3270	5250	VT
Yes	Yes	Yes

The **Start Host Notification** function begins the process by which your EHLLAPI application program determines if the host presentation space or OIA have been updated.

After using this function, your application program can use the **Query Host Update** (24) function to determine when a host event has occurred.

Prerequisite Calls

There are no prerequisite calls for this function.

Call Parameters

	Standard Interface	Enhanced Interface
Function Number	Must be 23	
Data String	Preallocated string; see the following table	
Length	6 or 7 implied	16
PS Position	NA	

The calling data string contains these elements:

Byte		Definition
Standard	Enhanced	
1	1	One of the following values: <ul style="list-style-type: none"> • A 1-character presentation space short name (PSID) • A blank or null indicating a request for the host-connected host presentation space
	2–4	Reserved.
2	5	One of the following values: <ul style="list-style-type: none"> • The character B asking for notification of both host presentation space and OIA updates. • The character O asking for notification of only OIA updates. • The character P asking for notification of only host presentation space updates. • The character A requesting the asynchronous mode of the notification. When A is specified, position 9–12 returns the address of an event object. The event selection character B, O, or P must be placed in position 13. • The character M requesting the asynchronous message mode of the notification. <p>When M is specified, the event selection character B, O, or P must be placed in position 13 (7 for 16-bit).</p> • E The character E asking for notification of completion during a printer session.
	6–8	Reserved.
3–4	9–12	When M is specified in position 5 (2 for 16-bit), the window handle of the window that receives the message should be set. The message is a return value of RegisterWindowMessage (PCSHLL) (not equal 0).

Byte		Definition
7	13	One of the following values if position 5 (2 for 16-bit) is A or M: <ul style="list-style-type: none"> • The character B asking for notification of both host presentation space and OIA updates • The character O asking for notification of only OIA updates • The character P asking for notification of only host presentation update.
	14–16	Reserved.

Return Parameters

This function returns a data string and a return code.

Data String:

If A (asynchronous mode of notification) is specified in position 5 and the function is completed successfully, the following data string is returned:

Byte		Definition
Standard	Enhanced	
1	1	A 1-character presentation space short name (PSID).
	2–8	Reserved.
	9–12	4-byte value in which the event object address is returned by EHLLAPI. The application can wait for this event object (32-bit only).

Data String:

If M (asynchronous message mode) is specified in position 5 (2 for standard interface) and the function is completed successfully, the following data string is returned:

Byte		Definition
Standard	Enhanced	
1	1	A 1-character presentation space short name (PSID)
	2–8	Reserved
3–4	9–10	Task ID of asynchronous message mode



Note: If OIA or presentation space is updated, an application window receives a message. The message is a return value of RegisterWindowMessage (PCSHLL). The wParam parameter contains the Task ID returned by the function call. HIWORD of lParam contains Return Code 21 (shows the OIA is updated), 22 (shows the host



presentation space is updated), or 23 (shows both the OIA and the host presentation space are updated), and LOWORD of IPParam parameter contains function number 23.

Return Code:

The following codes are defined:

Return Code	Definition
0	The Start Host Notification function was successful.
1	An incorrect host presentation space was specified.
2	An error was made in designating parameters.
9	A system error was encountered.

Notes on Using This Function

1. An application program can issue this function for multiple host sessions. The **Pause** (18) function can notify the application when one or more host sessions (PS, OIA, or both of them) are updated. The **Query Host Update** (24) function can be used to determine whether a PS, OIA, or both of them have been updated.
2. If the application chooses the asynchronous option, it can wait for the returned event object or semaphore to determine when a host event has occurred.
3. The event object or semaphore is initially in a non-signaled state and is signaled each time an appropriate event occurs. To receive notification for multiple events, the application must put the event object into the non-signaled state each time it has been signaled using either the **ResetEvent** or the **Query Host Update** (24) function.
4. An application cannot request Start Host Notification more than once with the same options.
5. This is not an exclusive call. Multiple applications can request this function for the same short session ID.

Start Keystroke Intercept (50)

3270	5250	VT
Yes	Yes	Yes

The **Start Keystroke Intercept** function allows a workstation application to filter any keystrokes sent to a session by a terminal operator. After a call to this function, keystrokes are intercepted and saved until the keystroke queue overflows or until the **Stop Keystroke Intercept** (53) function or **Reset System** (21) function is called. The intercepted keystrokes can be:

- Received through the **Get Key** (51) function and sent to the same or another session with the **Send Key** (3) function
- Accepted or rejected through the **Post Intercept Status** (52) function
- Replaced by other keystrokes with the **Send Key** (3) function
- Used to trigger other processes

Prerequisite Calls

There are no prerequisite calls for this function.

Call Parameters

	Standard Interface	Enhanced Interface
Function Number	Must be 50	
Data String	See the following table	
Length	Keystroke buffer size EHLLAPI allocates 32 bytes minimum for this buffer.	
PS Position	NA	

The calling data string contains:

Byte		Definition
Standard	Enhanced	
1	1	One of the following values: <ul style="list-style-type: none"> • A specific host presentation space short name (PSID) • A blank or null indicating a request for the host-connected host presentation space
	2–4	Reserved.
2	5	An option code character: <ul style="list-style-type: none"> • D for AID keystrokes only. • L for all keystrokes. • E for edit keys and all keystrokes (Available in Enhanced mode only) • M for requesting the asynchronous message mode of the notification (Windows only). <p>When M is specified, a code character D, or L, or E (Enhanced Mode) must be placed in position 13 (7 for 16-bit).</p> <p>Prerequisite: keyboard keys must be mapped to edit functions, e.g. Ctrl+C mapped to edit copy function. See Table 12: Mnemonics with @S (Shift), @W (Edit) and @ Alphabetic Characters on page 135 for edit functions supported.</p>
	6–8	Reserved.
3–4	9–12	When M is specified in position 5 (2 for 16-bit), the window handle of the window that receives the message should be set. The message is a return value of RegisterWindowMessage (PCSHLL) (not equal 0).

Byte		Definition
7	13	One of the following values if position 5 (2 for 16-bit) is M: <ul style="list-style-type: none"> • D for AID keystrokes only. • L for all keystrokes. • E for edit keys and all keystrokes. (Available in Enhanced mode only.)
	14–16	Reserved.

Data String:

If M (asynchronous message mode) is specified in position 5 (2 for standard interface) and the function is completed successfully, the following data string is returned:

Byte		Definition
Standard	Enhanced	
1	1	A 1-character presentation space short name (PSID)
	2–8	Reserved
3–4	9–10	Task ID of asynchronous message mode



Note: If a user sends keystrokes to a session, an application window receives a message. The message is a return value of RegisterWindowMessage (PCSHLL). The wParam parameter contains the Task ID returned by the function call. HIWORD of lParam parameter contains return code 0, which shows that the function was successful, and LOWORD of lParam parameter contains function number 50.

Return Parameters

Return Code	Explanation
0	The Start Keystroke Intercept function was successful.
1	An incorrect presentation space was specified.
2	An incorrect option was specified.
4	The execution of the function was inhibited because the target presentation space was busy.
9	A system error was encountered. Release is being used.

Notes on Using This Function

1. If a return code of 31 occurs for the **Get Key** (51) function, either:
 - Increase the value of the calling length parameter for this function, or
 - Execute the **Get Key** (51) function more frequently.

An intercepted keystroke occupies 3 bytes in the buffer. The next intercepted keystroke is placed in the adjacent 3 bytes. When the **Get Key** (51) function retrieves a keystroke (first-in first-out, or FIFO), the 3 bytes that it occupied are made available for another keystroke. By increasing the size of the buffer or the rate at which keystrokes are retrieved from the buffer, you can eliminate buffer overflow.

In the PC/3270, another way to eliminate return code 31 is to operate the PC/3270 emulator in the resume mode.

2. If option code D is provided, EHLLAPI writes intercepted non-AID keys to the presentation space to which they were originally intended, and returns only AID keys to the application.
3. Call the **Stop Keystroke Intercept** (53) function before exiting your EHLLAPI application. Otherwise, keystroke interception remains enabled with unpredictable results.

Start Playing Macro (110)

3270	5250	VT
Yes	Yes	Yes

The **Start Playing Macro** function invokes a macro. The macro will be executed in the connected session.



Note: This macro must exist in the Z and I Emulator for Windows user-class application data directory and no extension should be specified in the function call for the macro name.

Prerequisite Calls

Connect Presentation Space (1)

Call Parameters

	Standard Interface
Function Number	Must be 110
Data String	See the following table
Length	Length of macro name, plus 3
PS Position	NA

Byte		Definition
Standard	Enhanced	
1-2		Reserved
3-n		Null terminated macro name

Return Parameters

Return Code	Explanation
0	The Start Playing Macro function was successful.
1	The programs is not connected to a host session.
2	An error was made in specifying parameters.
9	A system error was encountered.

Stop Close Intercept (43)

3270	5250	VT
Yes	Yes	Yes

The **Stop Close Intercept** function allows the application to turn off the **Start Close Intercept** (41) function. After the application has issued the **Stop Close Intercept** function, subsequent close requests result in a normal stop sent to the logical terminal session.

Prerequisite Calls

Start Close Intercept (41)

Call Parameters

	Standard Interface	Enhanced Interface
Function Number	Must be 43	
Data String	1-character short session ID of the host presentation space	
Length	1	Must be 4
PS Position	NA	

The calling data string can contain:

Byte		Definition
Standard	Enhanced	
1	1	A 1-character presentation space short name (PSID)
	2-4	Reserved

Return Parameters

Return Code	Explanation
0	The Stop Close Intercept function was successful.
1	An incorrect host presentation space was specified.
2	An error was made in specifying parameters.

Return Code	Explanation
8	No previous Start Close Intercept (41) function was issued.
9	A system error occurred.
12	The session stopped.

Stop Communication Notification (82)

3270	5250	VT
Yes	Yes	Yes

The **Stop Communication Notification** function disables the capability of the **Query Communication Event** (81) function to determine whether any communication events have occurred in the specified Session.

Prerequisite Calls

Start Communication Notification (80)

Call Parameters

	Enhanced Interface
Function Number	Must be 82
Data String	1-character short name of the host presentation space, or a blank or null indicating request for updates to the host-connected presentation space
Length	4 is implied
PSPosition	NA

The calling data structure contains these elements:

Byte	Definition
1	A 1-character presentation space short name (PSID)
2-4	Reserved

Return Parameters

Return Code	Definition
0	The function was successful
1	An incorrect PSID was specified
8	No prior call to Start Communication Notification (80) function was called for the PSID
9	A system error was encountered

Stop Host Notification (25)

3270	5250	VT
Yes	Yes	Yes

The **Stop Host Notification** function disables the capability of the **Query Host Update** (24) function to determine if the host presentation space or OIA has been updated. This function also stops host events from affecting the **Pause** (18) function.

Prerequisite Calls

Start Host Notification (23)

Call Parameters

	Standard Interface	Enhanced Interface
Function Number	Must be 121	
Data String	See the following note	
Length	1 is implied	Must be 4
PS Position	NA	

The calling data string can contain:

Byte		Definition
Standard	Enhanced	
1	1	A 1-character presentation space short name (PSID)
	2-4	Reserved



Note: 1-character short name of the target presentation space ID, or a blank or a null to indicate a request for the host-connected presentation space.

Return Parameters

Return Code	Definition
0	The Stop Host Notification function was successful.
1	An incorrect host presentation space was specified.
8	No previous Start Host Notification (23) function was issued.
9	A system error was encountered.

Stop Keystroke Intercept (53)

3270	5250	VT
Yes	Yes	Yes

The **Stop Keystroke Intercept** function ends your application program's ability to intercept keystrokes.

Prerequisite Calls

Start Keystroke Intercept (50)

Call Parameters

	Standard Interface	Enhanced Interface
Function Number	Must be 53	
Data String	Short name of the target presentation space (PSID)	
Length	1 is implied	Must be 4
PS Position	NA	

The calling data string can contain:

Byte		Definition
Standard	Enhanced	
1	1	A 1-character presentation space short name (PSID)
	2–4	Reserved

Return Parameters

Return Code	Explanation
0	The Stop Keystroke Intercept function was successful.
1	An incorrect presentation space was specified.
8	No prior Start Keystroke Intercept (50) function was called for this presentation space.
9	A system error was encountered.

Wait (4)

3270	5250	VT
Yes	Yes	Yes

The **Wait** function checks the status of the host-connected presentation space. If the session is waiting for a host response (indicated by XCLOCK (X []) or XSYSTEM), the **Wait** function causes EHLLAPI to wait up to 1 minute to see if the condition clears.

Prerequisite Calls

Connect Presentation Space (1)

Call Parameters

	Standard Interface	Enhanced Interface
Function Number	Must be 4	
Data String	NA	
Length	NA	
PS Position	NA	

Return Parameters

Return Code	Definition
0	The keyboard is unlocked and ready for input.
1	Your application program is not connected to a valid session.
4	Timeout while still in XCLOCK (X <input type="checkbox"/>) or XSYSTEM.
5	The keyboard is locked.
9	A system error was encountered.

Notes on Using This Function

1. The **Wait** function is used to give host requests like those made by the **Send Key** (3) function the time required to be completed. Using the **Set Session Parameters** (9) function, you can request the `TWAIT`, `LWAIT`, or the `NWAIT` option. See item [12 on page 144](#).
2. You can use this function to see if the host OIA is inhibited.
3. The **Wait** function is satisfied by the host unlocking the keyboard. Therefore, a return code of 0 does not necessarily mean that the transaction has been completed. To verify completion of the transaction, you should use the **Search Field** (30) function or **Search Presentation Space** (6) function combined with the **Wait** function to look for expected keyword prompts.

Window Status (104)

3270	5250	VT
Yes	Yes	Yes

The **Window Status** function allows the application to query or change a window's presentation space size, location, or visible state.


Prerequisite Calls

Connect Window Services (101)

Call Parameters

	Standard Interface	Enhanced Interface
Function Number	Must be 104	
Data String	See the following table	
Length	16 or 20	24 or 28
PS Position	NA	

The calling data string can contain:

Byte		Definition
Standard	Enhanced	
1	1	A 1-character presentation space short name (PSID)
	2–4	Reserved
2	5	<p>A request option value, select one of the following values:</p> <ul style="list-style-type: none"> • X'01' for set status <div style="border: 1px solid black; padding: 5px; margin: 10px 0;">  Note: When the session is embedded In-Place in a compound OLE document, the set form of this function (byte 5 = X'01') always returns 0 but has no effect. </div> <ul style="list-style-type: none"> • X'02' for query for status • X'03' for query for extended status
	6	Reserved

If the request option value is X'01' (set status):

Byte		Definition
Standard	Enhanced	
3–4	7–8	<p>A 16- or 32-bit word containing the status set bits if the request option is 1 (set status). The following codes are valid return values if the request option is set status:</p> <p>X'0001'</p> <p>Change the window size. (Not valid with minimize, maximize, restore, or move.)</p>

Byte		Definition
		<p>X'0002'</p> <p>Move the window. (Not valid with minimize, maximize, size, or restore.)</p> <p>X'0004'</p> <p>ZORDER window replacement.</p> <p>X'0008'</p> <p>Set the window to visible.</p> <p>X'0010'</p> <p>Set the window to invisible.</p> <p>X'0080'</p> <p>Activate the window. (Sets focus to window and places it in the foreground unless ZORDER is specified. In this case, the ZORDER placement is used.)</p> <p>X'0100'</p> <p>Deactivate the window. (Deactivates the window and makes the window the bottom window unless ZORDER is also specified. In this case, the ZORDER placement is used.)</p> <p>X'0400'</p> <p>Set the window to minimized. (Not valid with maximize, restore, size, or move.)</p> <p>X'0800'</p> <p>Set the window to maximized. (Not valid with minimize, restore, size, or move.)</p> <p>X'1000'</p> <p>Restore the window. (Not valid with minimize, maximize, size, or move.)</p>
5–6	9–12	A 16- or 32-bit word containing the X window position coordinate. (Ignored if the move option is not set.)
7–8	13–16	A 16- or 32-bit word containing the Y window position coordinate. (Ignored if the move option is not set.)
9–10	17–20	A 16- or 32-bit word containing the X window size in device units. (Ignored if the size option is not set.)
11–12	21–24	A 16- or 32-bit word containing the Y window size in device units. (Ignored if the size option is not set.)

Byte		Definition
13–16	25–28	<p>A 16- or 32-bit word containing a window handle for relative window placement. These two words are only for the set option. (Ignored if the ZORDER option is not set.) Valid values are as follows:</p> <p>X'00000003' Place in front of all sibling windows. X'00000004' Place behind all sibling windows.</p>

If the request option value is X'02' (query for status):

Byte		Definition
Standard	Enhanced	
3–4	7–8	<p>A 16- or 32-bit word containing X'0000' if the request option is 2 (query for status). The following codes are possible return values if the request option is query for status. More than one state is possible.</p> <p>X'0008'</p> <p>The window is visible.</p> <p>X'0010'</p> <p>The window is invisible.</p> <p>X'0080'</p> <p>The window is activated.</p> <p>X'0100'</p> <p>The window is deactivated.</p> <p>X'0400'</p> <p>The window is minimized.</p> <p>X'0800'</p> <p>The window is maximized.</p>
5–6	9–12	A 16- or 32-bit word containing the X window position coordinate. (Ignored if the move option is not set.)
7–8	13–16	A 16- or 32-bit word containing the Y window position coordinate. (Ignored if the move option is not set.)
9–10	17–20	A 16- or 32-bit word containing the X window size in device units. (Ignored if the size option is not set.)
11–12	21–24	A 16- or 32-bit word containing the Y window size in device units. (Ignored if the size option is not set.)
13–16	25–28	A 16- or 32-bit word containing a window handle for relative window placement. These two words are only for the set option. (Ignored if the ZORDER option is not set.) Valid values are as follows:

Byte		Definition
		X'00000003' Place in front of all sibling windows. X'00000004' Place behind all sibling windows.

If the request option value is X'03' (query for extended status):

Byte		Definition
Standard	Enhanced	
3–4	7–8	<p>A 16- or 32-bit word containing X'0000' if the request option is 3 (query for extended status). The following codes are possible return values if the request option is query for extended status. More than one state is possible.</p> <p>X'0008'</p> <p>The window is visible.</p> <p>X'0010'</p> <p>The window is invisible.</p> <p>X'0080'</p> <p>The window is activated.</p> <p>X'0100'</p> <p>The window is deactivated.</p> <p>X'0400'</p> <p>The window is minimized.</p> <p>X'0800'</p> <p>The window is maximized.</p>
5–6	9–10	A 16- or 32-bit word containing the current font size in the X-dimension. The value is in screen pels.
7–8	11–12	A 16- or 32-bit word containing the current font size in the Y-dimension. The value is in screen pels.
9–12	13–16	Reserved. This value is always zero.
13–14	17–18	A 16- or 32-bit word containing the row number of the first visible character of the presentation space. This value is usually one, unless the Fixed Size font option is in effect, and the window has been resized such that some of the presentation space is hidden.
15–16	19–20	A 16- or 32-bit word containing the column number of the first visible character of the presentation space.
17–20	21–24	A 16- or 32-bit word containing the presentation space window handle of the session.

Return Parameters

Return Code	Explanation
0	The Window Status function was successful.
1	The presentation space was not valid or not connected.
2	An incorrect option was specified.
9	A system error occurred.
12	The session stopped.

Notes on Using This Function

The logical terminal (LT) windows use character cells. When resizing the LT windows, the LT rounds the number to prevent character cell truncation. The requested size and position might be slightly different from what was requested. Follow the set option with a query option to determine the final Presentation Manager® window position and size. All x and y coordinate positions and sizes are in pels.

Write Structured Fields (127)

3270	5250	VT
Yes	No	No

The **Write Structured Fields** function allows an application to write structured field data to the host application. If the call specifies S (for Synchronous), the application does not receive control until the **Write Structured Fields** function is completed. If the call specifies A (for Asynchronous), the application receives control immediately after the call. If the call specifies M, the application receives control immediately after the call. The application may wait for the message. In any case (S, A or M), the application provides the buffer address in which data to the host is to be placed.

For a successful asynchronous completion of this function, the following statements apply:

The return code field in the parameter list might not contain the results of the requested I/O. If the return code is not 0, then the request failed. The application must take the appropriate action based on the return code.

If the return code for this request is 0, the application must use the request ID returned with this function call to issue the **Get Request Completion** function call to determine the completion results of the function associated with the request ID. The **Get Request Completion** function call returns the following information:

1. Function request ID
2. Address of the data string from the asynchronous request
3. Length of the data string
4. Return code of the completed function

Prerequisite Calls

Connect for Structured Fields (120) Allocate Communication Buffer (123)

Call Parameters

	Standard Interface	Enhanced Interface
Function Number	Must be 127	
Data String	See the following table	
Length	8, 10, or 14	Must be 20
PS Position	NA	

The calling data string can contain:

Byte		Definition
Standard	Enhanced	
1	1	A 1-character presentation space short name (PSID).
	2–4	Reserved.
2	5	S or A or M S = Synchronous. Control is not returned to the application until the read is satisfied. A = Asynchronous. Control is returned immediately to the application, can wait for the event object. M = Asynchronous. Control is returned immediately to the application, can wait for the message.
	6	Reserved.
3–4	7–8	2-byte destination/origin ID.
5–8	9–12	4-byte address of the buffer from which the data is to be written. The buffer must be obtained using the Allocate Communications Buffer (123) function.
9–10	13–16	Reserved.
11–12	17–20	When "M" is specified in position 5 (2 for 16-bit), the window handle of the window that receives the message should be set, The message is a return value of RegisterWindowMessage ("PCSHLL") (not equal 0).
13–14		The data in these positions is ignored by EHLLAPI However, no error is caused if the migrating program has data in these positions. This data is accepted to provide compatibility with migrating applications.

Return Parameters

This function returns a data string and a return code.

Data String:

If A (asynchronous) is specified in position 5 (2 for standard interface) and the function is completed successfully, the following data string is returned:

Byte		Definition
9–10	13–14	2-byte Function Request ID. It is used by the Get Request Completion (125) function to determine the completion of this function call.
	15–16	Reserved.
	17–20	4-byte value in which the event object address is returned by EHLLAPI. The application can wait for this event object. When the event object is cleared, the application must issue the Get Request Completion (125) function call to get results of the Write Structured Fields request. (32-bit only).



Note: An event object is returned for each successful asynchronous request. The event object should not be used again. A new event object is returned for each request and is valid for only the duration of that request.

Data String:

If M (asynchronous message mode) is specified in position 5 (2 for standard interface) and the function is completed successfully, the following data string is returned:

Byte		Definition
9–10	13–14	2-byte Function Request ID. It is used by the Get Request Completion (125) function to determine the completion of this function call.
	15–16	Reserved.
11–12	17–18	Task ID of asynchronous message mode.
	19–20	Reserved.



Note: If the function is completed successfully, an application window receive a message. The message is a return value of RegisterWindowMessage (PCSHLL). The wParam parameter contains the Task ID returned by the function call. HIWORD of lParam parameter contains return code 0, which shows the function was successful, and LOWORD of lParam parameter contains function number 127.

Return Code:

The following codes are defined:

Return Code	Explanation
0	The Write Structured Fields function was successful.

Return Code	Explanation
1	A specified host presentation space short session ID was not valid or was not connected.
2	An error was made in specifying parameters.
9	A system error occurred.
11	Resource unavailable (memory unavailable).
34	The message sent inbound to the host was canceled.
35	An outbound transmission from the host was canceled.
36	Request rejected. Lost contact with the host.
37	Failed. The host is inbound disabled.

Notes on Using This Function

1. Return code 35 will be returned when the first **Read Structured Fields** or **Write Structured Fields** is requested after an outbound transmission from the host is canceled. Corrective action is the responsibility of the application.
2. Return code 36 requires that the application disconnect from the emulation program and then reconnect to reestablish communications with the host. Corrective action is the responsibility of the application.
3. Return code 37 will be returned if the host is inbound disabled.
4. The EHLLAPI allows for a maximum of 20 asynchronous requests per application to be outstanding. A return code for unavailable resources (RC=11) is returned if more than 20 asynchronous requests are attempted.

The structured field data format is as follows:

Offset	Length	Contents
0	1 word	X'0000'
2	1 word	m (message length: the number of bytes of data in the message, the number does not include the buffer header prefix, which contains 8 bytes) This value must be set by the application.
4	1 word	X'0000'
6	1 word	X'0000'
8	8 bytes	Length of the first (or only) structured field message.
10	1 byte	First nonlength byte of the structured field message.
		:
m+7	1 byte	Last byte in the structured field message.

Bytes 0 through 7 are the buffer header. These first 8 bytes are used by the emulation program. The user section of the buffer begins with offset 8. Bytes 8 and 9 contain the number of bytes in the first structured field (a structured field message can contain multiple structured fields) including 2 bytes for bytes 8 and 9. Bytes 8 through $m+7$ are used for the structured field message sent to the host.

Synchronous Requests

When **Write Structured Fields** is requested synchronously (the S option in the data string), control is returned to the application only after the request is satisfied. The application can assume:

- The return code is correct.
 - The data in the communications buffer (read buffer) is correct.
 - The host is no longer processing the **Write Structured Fields** request.
-

Asynchronous Requests

When **Write Structured Fields** is requested asynchronously (the A option in the data string), the application *cannot* assume:

- The return code is correct.
- The data in the communications buffer (write buffer) is correct.
- The host is no longer processing the **Write Structured Fields** request.

When requested asynchronously, EHLLAPI returns the following values:

- A 16-bit Request ID in positions 13–14 (9–10 for standard interface) of the data string
- The address of a event object in positions 17–20 of the data string.

These are used to complete the asynchronous **Write Structured Fields** call.

The following steps must be completed to determine the outcome of an asynchronous **Write Structured Fields** function call:

- If the EHLLAPI return code is not zero, the request failed. No asynchronous request has been made. The application must take appropriate actions before attempting the call again.
 - If the return code is zero, the application should wait until the event object is in the signaled state by using the **Get Request Completion** (125) function. The event object **Get Request Completion** (125) function) and should not be reused. The event object is valid only for the duration of the **Write Structured Fields** function call through the completion of the **Get Request Completion** (125) function call.
 - Once the event object is in the signaled state use the returned 16-bit Request ID as the Request ID parameter in a call to the **Get Request Completion** (125) function. The data string returned from the **Get Request Completion** (125) function call contains the final return code of the **Write Structured Fields** function call.
-

Asynchronous Requests

When **Write Structured Fields** is requested asynchronously (the M option in the data string), the application cannot assume:

- The return code is correct
- The data in the communications buffer (write buffer) is correct
- The host is no longer processing the **Write Structured Fields** request

When requested asynchronously with the M option, EHLLAPI returns the following values:

- A 16-bit request ID in positions 13–14 (9–10 for standard interface) of the data string
- Task ID of asynchronous message mode in position 17–18 (11–12 for standard interface)

These are used to complete the asynchronous **Write Structured Fields** call.

Chapter 4. WinHLLAPI Extension Functions

This chapter describes the extension functions provided when using WinHLLAPI programming support.

Summary of WinHLLAPI Functions

The following WinHLLAPI functions are available for 3270, 5250, and VT:

- [Wait \(4\) on page 176](#)
 - [Start Host Notification \(23\) on page 177](#)
 - [Start Close Intercept \(41\) on page 178](#)
 - [Start Keystroke Intercept \(50\) on page 179](#)
 - [Send File \(90\) on page 180](#)
 - [Receive File \(91\) on page 182](#)
-

WinHLLAPI Asynchronous Functions

The following sections describe the WinHLLAPI asynchronous functions.

WinHLLAPIAsync

This entry point is used for six WinHLLAPI functions that often take a long time to complete. With WinHLLAPIAsync, the function will be launched asynchronously and will not interfere with the continued progression of the calling application. These functions are: **Wait** (04), **Start Host Notify** (23), **Start Close Intercept** (41), **Start Keystroke Intercept** (50), **Send File** (90), and **Receive File** (91), and are described in [WinHLLAPI Extension Functions on page 175](#).

HANDLE WinHLLAPIAsync (HWINH hWnd, LPWORD *lpnFunction*, LPBYTE *lpData*, LPWORD *lpnLength*, LPWORD *lpnRetC*)*

The parameter list is the same as WinHLLAPI except a window handle is required before the function number. Since the function operates asynchronously, its completion is signaled by a registered message. The window handle is required as the target of the message.

There are two messages that must be registered by the WinHLLAPI application through calls to **RegisterWindowsMessage()** with the strings **WinHLLAPIAsync**(for all functions except 90 and 91) and **WinHLLAPIAsyncFileTransfer** (for functions 90 and 91). The standard format is as follows:

WPARAM

contains the Task Handle returned by the original function call.

LPARAM

the high word contains the error code and the low word contains the original function number.

Wait (4)

This function determines whether the Host session is in an inhibited state. If, for some reason, the session is in an inhibited state, this function will signal your application with a message when either the inhibited state expires or your wait period has expired. The amount of time to wait is set with the **Set Session Parameters (9)** function.

Prerequisite Functions

Connect Presentation Space (1)

WinHLLAPIAsync(*hWnd*, *lpwFunction*, *lpbyString*, *lpwLength*, *lpwReturnCode*)

Call Parameters

Parameter	Description
<i>Data String</i>	NA
<i>Data Length</i>	NA
<i>PS Position</i>	NA

Return Codes

Code	Description
WHLLOK	The PS is uninhibited and ready for input.
WHLLNOTCONNECTED	Your WinHLLAPI application is not connected to a valid host session.
WHLLPSBUSY	Function timed out while still inhibited.
WHLLNHIBITED	The PS is inhibited.
SHLLSYSERROR	The function failed due to a system error.
WHLLCANCEL	The asynchronous function was cancelled.

Remarks

Asynchronous Wait is used to notify the calling application when the inhibited state of the PS is expired. When inhibited state has expired, this version of **Wait** will post a **WinHLLAPIAsync** message to the window specified by the *hWnd*. The session options **TWAIT**, **LWAIT**, and **NWAIT** affect the length of time that this function will wait. See [Set Session Parameters \(9\) on page 139](#) for details on these session options.



Note: If **NWAIT** is specified in the session parameters and the application registers using revision 1.1 of the WinHLLAPI implementation, the **WinHLLAPIAsync** call will work the same as the **WinHLLAPI** call and not



send a message. If revision 1.0 is being used then **Wait** will return a message immediately with the inhibited status of the PS.

Start Host Notification (23)

This function enables you to notify your WinHLLAPI application of changes in the Host Session Presentation Space (PS) or Operation Information Area (OIA).

Prerequisite Functions

There are no prerequisite functions for this function.

WinHLLAPIAsync (*hWnd*, *lpwFunction*, *lpbyString*, *lpwLength*, *lpwReturnCode*)

Call Parameters

Parameter	Description
<i>Data String</i>	<p>A 7-byte string in the following format:</p> <p>Byte 1</p> <p>Short name session ID of the desired Host session, or space or null for the current Host session.</p> <p>Byte 2</p> <p>Notification mode. "P" for presentation space update only, "O" for OIA update only, "B" for both presentation space and OIA updates. When calling WinHLLAPIAsync, this position can be "A".</p> <p>Byte 3-6</p> <p>Not used. Provided for compatibility with older applications.</p> <p>Byte 7</p> <p>Reserved or replaced with one of the following if using WinHLLAPIAsync and A in byte 2: P for presentation space update only, O for OIA update only; and B for both presentation space and OIA updates.</p>
<i>Data Length</i>	Length of Host event buffer (256 recommended).
<i>PS Position</i>	NA

Return Parameters

Parameter	Description
<i>Data String</i>	Same as <i>Data String</i> on the call.

Return Codes

Code	Description
WHLLOK	Host notification enabled.
WHLLNOTCONNECTED	The specified Host session is invalid.
WHLLPARAMETERERROR	One of more parameters are invalid.
WHLLSYSERROR	The function failed due to a system error.
WHLLCANCEL	The asynchronous function was cancelled.

Remarks

Once enabled, Host notification is enabled until you call **Stop Host Notification (25)** or **WinHLLAPICancelAsyncRequest()**. The function initiates host notification and immediately returns control to your Windows HLLAPI application. This frees your application to perform other tasks while waiting for host updates. When an update occurs, the function will notify the window specified by *hWnd* with the registered message **WinHLLAPIAsync**.

Start Close Intercept (41)

This function intercepts user requests to close Z and I Emulator for Windows.

Prerequisite Functions

There are no prerequisite functions for this function.

WinHLLAPIAsync (*hWnd*, *lpwFunction*, *lpbyString*, *lpwLength*, *lpwReturnCode*)

Call Parameters

Parameter	Description
<i>Data String</i>	A 5-byte string for returned semaphore address. The first byte is the session short name of the session to query, or space or null for the current session.
<i>Data Length</i>	Must be specified.
<i>PS Position</i>	NA

Return Parameters

Parameter	Description
<i>Data String</i>	<p>A 5-byte string with the following format:</p> <p>Byte 1</p> <p>Session short name, or space or null for the current session</p> <p>Bytes 2-5</p> <p>Semaphore address.</p>

Return Code

Code	Description
WHLLOK	The function was successful.
WHLLNOTCONNECTED	An invalid presentation space was specified.
WHLLPARAMETERERROR	An invalid option was specified.
WHLLSYSERROR	The function failed due to a system error.
WHLLCANCEL	The asynchronous function was cancelled.

Remarks

Once enabled, Host notification remains enabled until you call **Stop Close Intercept (43)** or **WinHLLAPICancelAsyncRequest ()**. Initially, the semaphore is set. After using this function, close requests from the user are discarded and the semaphore is cleared.

The function initiates close intercept and immediately returns control to your Windows HLLAPI application. This frees your application to perform other tasks while waiting for close requests. When a close request occurs, the function will notify the window specified by *hWnd* with the registered message **WinHLLAPIAsync**.

Start Keystroke Intercept (50)

This function intercepts keystrokes sent to a session by the user.

Prerequisite Functions

There are no prerequisite functions for this function.

WinHLLAPIAsync (*hWnd*, *lpwFunction*, *lpbyString*, *lpwLength*, *lpwReturnCode*)

Call Parameters

Parameter	Description
<i>Data String</i>	A 6-byte string in the following format: Byte 1 Session short name, or space or null for the current Host session. Byte 2 Keystroke intercept code. "D" causes only AID keystrokes to be intercepted; "L" causes all keystrokes to be intercepted. Bytes 3-6 Reserved
<i>Data Length</i>	Variable (256 is recommended)
<i>PS Position</i>	NA

Return Code

Code	Description
WHLLOK	Keystroke intercept has been initiated.
WHLLNOTCONNECTED	The Host session presentation space is invalid.
WHLLPARAMETERERROR	One or more parameters are invalid.
WHLLPSBUSY	Session is busy.
WHLLSYSERROR	Function failed due to a system error.
WHLLCANCEL	Asynchronous function was cancelled.

Remarks

The function initiates keystroke intercept and immediately returns control to your Windows HLLAPI application. This frees your application to perform other tasks while waiting for keystrokes. Once initiated, the function will post a **WinHLLAPIAsync** message to the window specified by *hWnd* whenever the user sends a key to the PS. After notification, the intercepted keystrokes can be handled in any way that is allowed by a normal EHLLAPI application. Take note that the keystroke buffer is of limited size so each keystroke should be handled and removed from the buffer.

Send File (90)

This function transfers a file from the PC to the Host.

Prerequisite Functions

There are no prerequisite functions for this function.

WinHLLAPIAsync (*hWnd*, *lpwFunction*, *lpbyString*, *lpwLength*, *lpwReturnCode*)

Call Parameters

Parameter	Description
<i>Data String</i>	SEND command parameters.
<i>Data Length</i>	Length of <i>Data String</i> . NA if session option EOT is specified.
<i>PS Position</i>	NA

Return Codes

Code	Description
WHLLOK	File transfer started successfully.
WHLLPARAMETERERROR	Parameter error or <i>Data Length</i> is zero or greater than 255.
WHLLFTXCOMPLETE	File transfer complete.
WHLLFTXSEGMENTED	Transfer is complete with segmented records.
WHLLSYSERROR	The function failed due to a system error.
WHLLTRANSABORTED	File transfer aborted, either due to the user clicking the cancel button or because the timeout period has elapsed.
WHLLFILENOTFOUND	PC file not found.
WHLLFTXCOMPLETECICS	File transfer was successful (transfer to CICS).
WHLLACCESSDENIED	Access denied to PC file.
WHLLMEMORY	Insufficient memory.
WHLLINVALIDENVIRONMENT	Invalid environment.

Remarks

Only one file transfer operation is supported per connected Host session.

The function initiates the file transfer and immediately returns control to your Windows HLLAPI application. This frees your application to perform other tasks while the file transfer is occurring. Once initiated the function will regularly post **WinHLLAPIAsyncFileTransfer** messages to the window specified by *hWnd*. These messages will notify the WinHLLAPI application of the status of the transfer and send a final message when the transfer is complete.

wParm

Is the status indicator: the high byte contains the Session ID, the low byte contains the status. If the low byte is zero, the file transfer is still in progress. If the low byte is one, the file transfer has completed.

IParm

If the low byte of *wParm* is zero (in progress), *IParm* is the number of bytes transferred. If the low byte *wParm* is one (completed), *IParm* is the completion code.

Receive File (91)

This function transfers a file from the PC to the Host.

Prerequisite Functions

There are no prerequisite functions for this function.

WinHLLAPIAsync (*hWnd*, *lpwFunction*, *lpbyString*, *lpwLength*, *lpwReturnCode*)

Call Parameters

Parameter	Description
<i>Data String</i>	RECEIVE command parameters.
<i>Data Length</i>	Length of <i>Data String</i> . NA if session option EOT is specified.
<i>PS Position</i>	NA

Return Codes

Code	Description
WHLLOK	File transfer started successfully.
WHLLPARAMETERERROR	Parameter error or <i>Data Length</i> is zero or greater than 255.
WHLLFTXCOMPLETE	File transfer complete.
WHLLFTXSEGMENTED	Transfer is complete with segmented records.
WHLLSYSERROR	The function failed due to a system error.
WHLLTRANSABORTED	File transfer aborted, either due to the user clicking the cancel button or because the timeout period has elapsed.
WHLLFILENOTFOUND	PC file not found.
WHLLFTXCOMPLETECICS	File transfer was successful (transfer to CICS).
WHLLACCESSDENIED	Access denied to PC file.
WHLLMEMORY	Insufficient memory.
WHLLINVALIDENVIRONMENT	Invalid environment.

Remarks

Only one file transfer operation is supported per connected Host session.

The function initiates the file transfer and immediately returns control to your Windows HLLAPI application. This frees your application to perform other tasks while the file transfer is occurring. Once initiated the function will regularly post **WinHLLAPIAsyncFileTransfer** messages to the window specified by *hWnd*. These messages will notify the WinHLLAPI application of the status of the transfer and send a final message when the transfer is complete.

wParm

Is the status indicator: the high byte contains the Session ID, the low byte contains the status. If the low byte is zero, the file transfer is still in progress. If the low byte is one, the file transfer has completed.

lParm

If the low byte of *wParm* is zero (in progress), *lParm* is the number of bytes transferred. If the low byte *wParm* is one (completed), *lParm* is the completion code.

WinHLLAPICancelAsyncRequest

This function cancels an outstanding asynchronous function launched by a call to **WinHLLAPIAsync()**.

Syntax

```
int WinHLLAPICancelAsyncRequest (HANDLE hAsyncTask, WORD wFunction)
```

Parameters

hAsyncTask

The handle returned by WinHLLAPIAsync() when the function was initiated.

wFunction

The function number of the asynchronous task to cancel. Because this parameter is required for revision 1.1 but not in 1.0, it is optional.

With this function, any asynchronous task previously initiated by a call to WinHLLAPIAsync() may be canceled while still outstanding.

Returns

The return value indicates if the specified function was, in fact, canceled. If the function was canceled then the return value is WHLLOK (0). If the outstanding asynchronous function was not cancelled, one of the following codes will be returned.

WHLINVALID

hAsyncTask is not a valid task handle.

WHLLALREADY

The asynchronous task specified by `hAsyncTask` has already completed.

Initialization and Termination Functions

The following section describes the initialization and termination functions of WinHLLAPI programming support.

WinHLLAPI Startup

This function is used to register the application with the WinHLLAPI implementation and should be called before any other call to the WinHLLAPI implementation. This implementation supports Versions 1.0 and 1.1 of the WinHLLAPI specification. The WinHLLAPI application should negotiate version compatibility with this function.

Syntax

```
int WinHLLAPIStartup(WORD wVersionRequired, LPWHLLAPIDATA lpData)
```

Parameters

wVersionRequired

This is the version required by the WinHLLAPI application. The low byte contains the major version number and the high byte contains the minor version (or revision) number.

lpData

This is a pointer to a WHLLAPIDATA structure which will receive the implementations version number and a string describing the WinHLLAPI implementation provider. The WHLLAPIDATA structure is defined as:

```
#define WHLLDESCRIPTION_LEN 127
typedef struct tagWHLLAPIDATA
{
    WORD wVersion;
    Char szDescription[WHLLDESCRIPTION_LEN + 1];
}WHLLAPIDATA, * PWHLLAPIDATA, FAR *LPWHLLAPIDATA;
```

Returns

The return value indicates success or failure of registering the WinHLLAPI application with the implementation. If registration was successful, the return value is `WHLLOK` (zero). Otherwise, it is one of the following:

WHLLSYSNOTREADY

Indicates that the underlying network subsystem is unavailable.

WHLLVERNOTSUPPORTED

Indicates that the version requested is not provided by this implementation. This implementation supports Versions 1.0 and 1.1 only.

WinHLLAPI Cleanup

The WinHLLAPI specification recommends that this function be used by the WinHLLAPI application to de-register from the WinHLLAPI implementation.

Syntax

BOOL WinHLLAPICleanup()

Returns

Returns TRUE if the unregistration was successful. Otherwise, it returns FALSE.

Blocking Routines

The following sections describe the blocking routines supported by WinHLLAPI programming.



Note: Although blocking routines are supported for WinHLLAPI compliance, use of them is not recommended. Use of the WinHLLAPIAsync functions are the recommended method for asynchronous processing.

WinHLLAPIIsBlocking

This function tells the calling WinHLLAPI application thread whether it is in the process of executing a blocking call. A blocking call is any synchronous function that takes a long time to execute and does not return until complete. There are five blocking calls in this implementation of WinHLLAPI. The blocking calls are: **Get Key (51)**, **Wait (4)**, **Pause (18)**, **Send File (90)**, and **Receive File (91)**.

Syntax

BOOL WinHLLAPIIsBlocking()

Returns

If the WinHLLAPI application thread is in the middle of a blocking call, the function returns TRUE, otherwise, it returns FALSE.

Remarks

Because the default blocking-hook allows messages to be processed during blocking calls, it is possible to call the blocking call again.

WinHLLAPISetBlockingHook

This function sets an application-defined procedure to be executed while waiting for the completion of a blocking call. A blocking call is any synchronous function that takes a long time to execute and does not return until complete. There are five blocking calls in this implementation of WinHLLAPI. The blocking calls are: **Get Key (51)**, **Wait (4)**, **Pause (18)**, **Send File (90)**, and **Receive File (91)**.

Syntax

FARPROC WinHLLAPISetBlockingHook(FARPROC *lpfnBlockingHook*)

Parameters

lpfnBlockingHook

This is a pointer to the new blocking procedure.

Description

The WinHLLAPI implementation has a default blocking procedure that consists of nothing more than a message handler. This default mechanism is shown in the following example:

```
BOOL    DefaultBlockingHook
{
    MSG msg;

    if (PeekMessage (&msg, NULL, 0, 0, xFPM_NOREMOVE))
    {
        if(msg.message == WM_QUIT)
        {
            return FALSE;
        }
        PeekMessage (&msg, NULL, 0, 0, PM_REMOVE);
        TranslateMessage (&msg);
        DispatchMessage (&msg);
    }
    return TRUE;
}
```

The blocking hook is implemented on a per-thread basis. A blocking hook set by this function will stay in effect for the thread until it is replaced by another call to **WinHLLAPISetBlockingHook()** or until the default is restored by a call to **WinHLLAPIUnhookBlockingHook()**.

The Blocking function must return **FALSE** if it receives a **WM_QUIT** message so WinHLLAPI can return control to the application to process the message and terminate gracefully. Otherwise, the function should return **TRUE**.

Returns

This function returns a pointer to the blocking function being replaced.

WinHLLAPIUnhookBlockingHook

This function restores the default blocking-hook for the calling thread.

Syntax

BOOL WinHLLAPIUnhookBlockingHook()

Returns

This function returns TRUE if the default blocking mechanism was successfully restored, otherwise it returns FALSE.

WinHLLAPICancelBlockingCall

This function cancels an executing blocking call in the *current thread*. A blocking call is any synchronous function that takes a long time to execute and does not return until complete. There are five blocking calls in this implementation of WinHLLAPI. The blocking calls are **Get Key** (51), **Wait** (4), **Pause** (18), **Send File** (90), and **Receive File** (91). If one of these is blocking calls are cancelled, the cancelled function will return WHLLCANCEL.

Syntax

int WinHLLAPICancelBlockingCall()

Returns

The return value indicates if the specified function was, in fact, canceled. If the function was canceled, then the return value is WHLLOK (0). If there are no outstanding blocking functions, then the following return code will be returned:

WHLLINVALID

Indicates that there is no blocking call currently executing.

Chapter 5. PCSAPI Functions

Z and I Emulator for Windows provides an API set, which is defined here and called *PCSAPI*. Whereas EHLLAPI is used to manage the interaction between a workstation application program and host systems after the session is established, the PCSAPI can be used to control the Z and I Emulator for Windows session itself.

How to Use PCSAPI

You can write application programs using the PCSAPI in C or C++. To develop a PCSAPI application, do the following:

1. Prepare source code and add the appropriate PCSAPI calls.
2. Include the header file PCSAPI.H in the application program.
3. Compile the source code.
4. Link the resultant .OBJ files with the appropriate object file or libraries.

You must also link it with the PCSAPI import library, PCSCALLS.LIB for 16-bit and PCSCAL32.LIB for 32-bit.

Page Layout Conventions

All PCSAPI function calls are presented in the same format so that you can quickly retrieve the information you need. The format is:

- Function Name
 - Function Type
 - Parameter Type and Description
 - Return Code
-

Function Type

“Function Type” shows the type of the function in the following format:

TYPE **FunctionName**(*TYPE Parameter1, ...*)

Parameter Type and Description

“Parameter Type and Description” lists the type and describes each of the parameters to be specified in the PCSAPI function call.

Return Code

“Return Code” lists the codes that must be received by your program after a call to the PCSAPI function.

pcsConnectSession

3270	5250	VT
Yes	Yes	Yes

The **pcsConnectSession** function starts the communications with a host session specified by the short session ID. The session must already be started. This call is equivalent to the **Communications → Connect** menu item on the emulator session panel.

Function Type

BOOL WINAPI pcsConnectSession(char cShortSessionID)

Parameter Type and Description

char cShortSessionID

Presentation space short session ID.

Return Code

Return Code	Meaning
TRUE	Function ended successfully.
FALSE	It means one of the following things: <ul style="list-style-type: none"> • The session has not started. • An incorrect session ID was specified. • Call failed.

pcsDisconnectSession

3270	5250	VT
Yes	Yes	Yes

The **pcsDisconnectSession** function stops the communications link with a host session specified by the short session ID. This only disconnects the link; it does not stop the session. This call is equivalent to the **Communications → Disconnect** menu item on the emulator session panel.

Function Type

BOOL WINAPI pcsDisconnectSession(char cShortSessionID)

Parameter Type and Description

char cShortSessionID

Presentation space short session ID.

Return Code

Return Code	Meaning
TRUE	Function ended successfully.
FALSE	It means one of the following things: <ul style="list-style-type: none">• The session has not started.• An incorrect session ID was specified.• Call failed.

pcsQueryConnectionInfo

3270	5250	VT
Yes	No	No

The **pcsQueryConnectionInfo** function returns information about the Telnet connection of the specified host session. The resulting information is returned into the buffer supplied by the application.

Function Type

BOOL WINAPI pcsQueryConnectionInfo(*char cShortSessionID*, *CONNECTIONINFO *ConnectionInfo*)

Parameter Type and Description

char cShortSessionID

Presentation space short session ID.

CONNECTIONINFO *ConnectionInfo

Pointer to a CONNECTIONINFO structure where the connection info data will be returned.

Return Code

Return Code	Meaning
TRUE	Function ended successfully.
FALSE	It means one of the following things:

Return Code	Meaning
	<ul style="list-style-type: none"> • The session has not started. • An incorrect session ID was specified. • The session specified was not a supported connection type for this API (not Telnet).

ConnectionInfo

The CONNECTIONINFO structure will be filled with the information about the host connection, consisting of the following information:

Structure	Information
Host name	States the name of the currently connected Telnet host.
LU name	States the LU name currently assigned.
Port number	States the host port number being used for the connection.
SSL indicator	Indicates a Secure Connection (1 = secure; 0 = not secure).



Note: This API is valid only with the 32-bit version of PCSAPI, and only works for Telnet connections.

Example

```
typedef struct_CONNECTIONINFO
{ //Description of a connection @WD06A
    char hostName[63];    //telnet host name      @WD06A
    char reserved[1];     //reserved          @wD06A
    int portNumber;       //host port number   @WD06A
    char luName[17];      //LU name            @WD06A
    char reserved2[3];    //reserved           @WD06A
    BOOL sslIndicator;    //Secure Connection  @WD06A
                        indicator
    char reserved3[256];  //reserved           @WD06A
}CONNECTIONINFO;
```

pcsQueryEmulatorStatus

3270	5250	VT
Yes	Yes	Yes

The **pcsQueryEmulatorStatus** function returns the status of the host session specified by the short session ID.

Function Type

ULONG WINAPI pcsQueryEmulatorStatus(char cShortSessionID)

Parameter Type and Description

char cShortSessionID

Presentation space short session ID.

Return Code

The return code value should be processed bit-significantly, that is, by either one of the following values or an ORed value out of the following values:

Return Code	Value	Meaning
PCS_SESSION_STARTED	0x00000001	Specified session has started. When this bit is off, the specified session has not started or an incorrect session ID was specified.
PCS_SESSION_ONLINE	0x00000002	Specified session is online (connected). When this bit is off, the specified session is offline (disconnected).
PCS_SESSION_API_ENABLED	0x00000004	API (EHLLAPI) is enabled on the specified session. If this bit is off, API is disabled on this session.

pcsQuerySessionList

3270	5250	VT
Yes	Yes	Yes

The **pcsQuerySessionList** function returns a list of all the current host sessions. The application must supply an array of SESSINFO structures as defined in the PCSAPI.H file, and a count of the number of elements in the array. This function fills in the structures with information about each session and returns the number of sessions found.

If the array has fewer elements than there are host sessions, then only the supplied elements of the array are filled in. The function always returns the actual number of sessions, even if the array is too small.

An application can call this function with zero array elements to determine how many sessions exist. A second call can then be made to obtain the session information.

Function Type

ULONG WINAPI pcsQuerySessionList(ULONG Count, SESSINFO *SessionList)

Parameter Type and Description

ULONG Count

Number of elements in the SessionList array.

SESSINFO *SessionList

Pointer to an array of SESSINFO structures as defined in PCSAPI.H.

Return Parameters

Return Code

Total number of Z and I Emulator for Windows sessions. This may be greater than or less than the Count parameter.

SessionList

The array of SESSINFO structures is filled with information about the host sessions. Sessions may be placed in the list in any order. Each SESSINFO structure contains the following fields (defined in PCSAPI32.H)

Name

A union of char and ULONG which contains the session ID (A–Z). In the current implementation of Z and I Emulator for Windows, only the lower byte (char) is used, the other bytes are returned as zero.

Status

A combination of bit flags which indicate the current status of the session. The flags (PCS_SESSION_*) are defined in the following table.

The status value should be processed bit-significantly, that is, by either one of the following values or an ORed value out of the following values:

Return Code	Meaning
PCS_SESSION_STARTED	The session is running. If this flag is not set, all others are undefined.
PCS_SESSION_ONLINE	The session has established a communications link to the host (this is, the session is connected).
PCS_SESSION_API_ENABLED	The session is enabled for programming APIs. If this flag is not set, the EHLLAPI and Host Access Class Library APIs cannot be used on this session.

Example

```
ULONG    NumSessions, i; // Session counters
SESSINFO *SessList;      // Array of session information structures
// Find out number of sessions that exist
```

```
NumSessions = pcsQuerySessionList (0,NULL);
if (NumSessions == 0) {
    printf("There are no sessions.");
    exit;
}

// Allocate array large enough for all sessions
SessList = (SESSINFO *)malloc(NumSessions * sizeof(SESSINFO));
memset(SessList, 0x00, NumSessions * sizeof(SESSINFO));

// Now read actual session info
pcsQuerySessionList(NumSessions, SessList);

for (i=0; i<NumSessions; i++) {
    if ((SessList[i].Status & PCS_SESSION_STARTED) &&
        (SessList[i].Status & PCS_SESSION_ONLINE)) {

        printf("Session %c is started and connected.",
            SessList[i].Name.ShortName);
    }
}

exit;
```

pcsQueryWorkstationProfile

3270	5250	VT
Yes	Yes	Yes

The **pcsQueryWorkstationProfile** function returns the workstation profile name that has been used to invoke the host session. To specify the host session, the short session ID must be used. The workstation profile name is copied to the work buffer supplied by the application.

Function Type

BOOL WINAPI **pcsQueryWorkstationProfile**(*char cShortSessionID*, *PSZ lpBuffer*)

Parameter Type and Description

char cShortSessionID

Presentation space short session ID.

PSZ lpBuffer

Work buffer to copy a null-terminated workstation profile name. The buffer must be large enough to contain a fully qualified file name.

Return Code

Return Code	Meaning
TRUE	Function ended successfully.
FALSE	It means one of the following things: <ul style="list-style-type: none"> • The session has not started. • An incorrect session ID was specified.

pcsSetLinkTimeout

3270	5250	VT
Yes	Yes	Yes

The **pcsSetLinkTimeout** function sets the idle timeout of a Telnet link which is SSCP owned. This function has no effect on non-TN connections or connections which are not in SSCP owned state. If the timeout value is set to zero the link will not time out. Otherwise the link will time out (disconnect) after being idle in SSCP-owned state for the number of minutes specified.

Function Prototype

ULONG WINAPI pcsSetLinkTimeout(char cShortSessionID, USHORT Timeout)

Parameter Type and Description

char cShortSessionID

Presentation space short session ID.

USHORT Timeout

Timeout value in minutes. A value of zero disables timeout.

Return Code

Return Code	Meaning
PCS_SUCCESSFUL	The function ended successfully.
PCS_SYSTEM_ERROR	A system error occurred.

pcsStartSession

3270	5250	VT
Yes	Yes	Yes

The **pcsStartSession** function starts a host session by using a specified workstation profile. A short session ID can also be specified.

Function Type

ULONG WINAPI pcsStartSession(*PSZ lpProfile, char cShortSessionID, USHORT fuCmdShow*)

Parameter Type and Description

PSZ lpProfile

Path and complete filename of the profile to load. Path is optional but complete filename must be specified (.ws extension is not assumed).

char cShortSessionID

Presentation space short session ID. Space or NULL indicates the next available session ID.

USHORT fuCmdShow

Specifies how the window is to be displayed. One of the following values from PCSAPI.H:

- PCS_HIDE
- PCS_SHOW
- PCS_MINIMIZE
- PCS_MAXIMIZE

Return Code

Return Code	Value	Meaning
PCS_SUCCESSFUL	0	The function ended successfully.
PCS_INVALID_ID	1	An incorrect session ID was specified.
PCS_USED_ID	2	The specified short session ID is already used.
PCS_INVALID_PROFILE	3	An error was made in specifying the workstation profile, or the window parameter was not valid.
PCS_SYSTEM_ERROR	9	A system error occurred.

pcsStopSession

3270	5250	VT
Yes	Yes	Yes

The **pcsStopSession** function stops a host session specified by the short session ID.

Function Type

BOOL WINAPI pcsStopSession(*char cShortSessionID, USHORT fuSaveProfile*)

Parameter Type and Description

char cShortSessionID

Presentation space short session ID.

USHORT fuSaveProfile

This parameter can be one of the following values:

fuSaveProfile	Value	Meaning
PCS_SAVE_AS_PROFILE	0	Save the profile as specified in the current profile.
PCS_SAVE_ON_EXIT	1	Save the profile on exit.
PCS_NOSAVE_ON_EXIT	2	Do not save the profile on exit.

Return Code

Return Code	Meaning
TRUE	The function ended successfully.
FALSE	It means one of the following things: <ul style="list-style-type: none"> • The session has not started. • An incorrect session ID was specified.

Page Setup Functions

The PCSAPI functions listed in this section enable you to control and retrieve the Z and I Emulator for Windows emulator session **Page Setup** settings.

Restrictions

If the following restrictions are not satisfied, the API will fail. The return code indicates the reason for the failure.

- The host session specified in the argument `cShortSessionID` should not be in PDT mode.
- The host session should not be printing when the API is invoked.
- The **File → Page Setup** dialog should not be in use.

Some members in the PAGEINFO structure might be valid or supported only for specific session types. If a restriction is not specified, then that member is valid or supported for the following session types:

- 3270 display
- 3270 printer
- 5250 display
- ASCII VT

5250 printer sessions are not supported.

pcsGetPageSettings

3270	5250	VT
Yes	Yes	Yes

The **pcsGetPageSettings** function retrieves the host session page settings values (similar to the **File → Page Setup** dialog settings). Only the settings in the **Text** tab of the dialog are supported.

Function Type

ULONG WINAPI pcsGetPageSettings(*char cShortSessionID*, *PAGEINFO * const pPageInfo*, *ULONG * const pErrorInfo*)

Parameter Type and Description

char cShortSessionID

Presentation space short session ID.

PAGEINFO * const pPageInfo

Pointer to PAGEINFO structure, where the page settings are returned.

nFlags

Combination of bit flags that indicates which members in the structure are valid. These flags can be used independently or by ORing them together to restore the property page (defined in PCSAPI32.H). The flags, along with the corresponding valid members in the structure, are as follows:

Flag

Valid members in the structure

PCS_PAGE_CPI

nCPI

PCS_PAGE_LPI

nLPI

PCS_PAGE_FACE_NAME

szFaceName

PCS_PAGE_MPL

nMPL

PCS_PAGE_MPP

nMPP

nCPI

The number of characters printed per inch.

LOWORD is the actual CPI value.

If Font CPI is configured in the session, HIWORD is 1. If Font CPI is not configured, HIWORD is 0.

nLPI

The number of lines printed per inch.

LOWORD is the actual LPI value.

If Font LPI is configured in the session, HIWORD is 1. If Font LPI is not configured, HIWORD is 0.

szFaceName

Face name of the printer font. This must be a null-terminated string.

nMPL

Maximum number of lines that can be printed per page.

This is also called MPL (Maximum Print Lines). Supported range is 1 to 255.

nMPP

Maximum number of characters that can be printed per line.

This is also called MPP (Maximum Print Position). Supported range is 1 to 255.

ULONG * const pErrorInfo

Not used. This must be set to NULL by the caller.

Return Code

Return Code	Value	Meaning
PCS_SUCCESSFUL	0	Function ended successfully.
PCS_INVALID_ID	1	Incorrect session ID was specified.
PCS_INVALID_SESS_TYPE	2	Not supported for the host session type.
PCS_DIALOG_IN_USE	3	Failed because the host session Page Setup or Printer Setup dialog was in use.

Return Code	Value	Meaning
PCS_PRINTING	4	Page settings cannot be obtained because host session was printing.
PCS_PDT_MODE	5	Page settings cannot be obtained because host session is in PDT mode.
PCS_SYSTEM_ERROR	9	A system error occurred.

Example

```

{
    ULONG Rc = 0;
    PAGEINFO *PageInfo;

    PageInfo = (PAGEINFO *) malloc(sizeof(PAGEINFO));
    memset(PageInfo, 0, sizeof(PAGEINFO));

    PageInfo->nFlags = PCS_PAGE_CPI | PCS_PAGE_LPI | PCS_PAGE_FACE_NAME |
                     PCS_PAGE_MPL | PCS_PAGE_MPP;

    Rc = pcsGetPageSettings('A', PageInfo, NULL);

    if (Rc == PCS_SUCCESSFUL) {
        printf("CPI = %d,\n",
              LPI = %d,\n",
              FaceName = %s,\n",
              MPL = %d,\n",
              MPP = %d\n",
              LOWORD(PageInfo->nCPI),
              LOWORD(PageInfo->nLPI),
              PageInfo->szFaceName,
              PageInfo->nMPL,
              PageInfo->nMPP);

        if (HIWORD(PageInfo->nCPI))
            printf("FontCPI\n");
        else
            printf("No FontCPI\n");

        if (HIWORD(PageInfo->nLPI))
            printf("FontLPI\n");
        else
            printf("No FontLPI\n");
    } else
        printf("Failure. Return code = %d\n", Rc);
    free(PageInfo);
}

```

pcsRestorePageDefaults

3270	5250	VT
Yes	Yes	Yes

The **pcsRestorePageDefaults** function restores the system default values of the Page Setup property pages defined in the **nFlags** field. This is equivalent to clicking **Default** in the property pages of the **File → Page Setup** dialog. Only the settings in the **Text** tab are supported.

Function Type

ULONG WINAPI pcsRestorePageDefaults(*char cShortSessionID, ULONG nFlags*)

Parameter Type and Description

char cShortSessionID

Presentation space short session ID.

ULONG nFlags

The following flag describes the name of the specified **Page Setup** dialog property page. This flag can be bitwise ORed to restore the property page (defined in PCSAPI32.H).

PCS_PAGE_TEXT

This flag describes the Text property page. This is the only property page currently supported.

Return Code

Return Code	Value	Meaning
PCS_SUCCESSFUL	0	Function ended successfully.
PCS_INVALID_ID	1	Incorrect session ID was specified.
PCS_INVALID_SESS_TYPE	2	The nFlags parameter has one or more options that are not valid for the host session type. No settings were restored.
PCS_DIALOG_IN_USE	3	Failed because the host session Page Setup or Printer Setup dialog was in use.
PCS_PRINTING	4	Page settings cannot be changed because host session was printing.
PCS_PDT_MODE	5	Page settings cannot be changed because host session is in PDT mode.
PCS_SYSTEM_ERROR	9	A system error occurred.

Example

```
{
    ULONG Rc = 0;

    Rc = pcsRestorePageDefaults('A', PCS_PAGE_TEXT);

    if (Rc != PCS_SUCCESSFUL)
```

```
printf("Failure. Return code = %d\n", Rc);
}
```

pcsSetPageSettings

3270	5250	VT
Yes	Yes	Yes

The **pcsSetPageSettings** function sets the host session page settings. This is similar to configuring the **File → Page Setup** dialog settings. Only the settings in the **Text** tab are supported.



Note:

1. CPI, LPI, and FontSize are dependent on the FaceName configured in the host session. If this API is used to set CPI, LPI, FontSize, and FaceName together, FaceName is set first, then the dependent properties.
2. If this API is used to set FaceName and the dependent properties in separate invocations, set FaceName first, then set CPI, LPI and FontSize. Otherwise, each time FaceName is set, query CPI, LPI and FontSize and ensure that they have the desired values.
3. If CPI, LPI, or FontSize are set before FaceName, then different values for CPI, LPI, or FontSize might be configured in the host session. This might occur if the current CPI, LPI, or FontSize values are not valid for the new FaceName set.

Function Type

ULONG WINAPI pcsSetPageSettings(*char cShortSessionID, const PAGEINFO * const pPageInfo, ULONG * const pErrorInfo*)

Parameter Type and Description

char cShortSessionID

Presentation space short session ID.

const PAGEINFO * const pPageInfo

Pointer to PAGEINFO structure, where the page settings are mentioned.

nFlags

Combination of bit flags that indicates which members in the structure are valid. These flags can be used independently or by ORing them together to restore the property page (defined in PCSAPI32.H). The flags, along with the corresponding valid members in the structure, are as follows:

Flag

Valid members in the structure**PCS_PAGE_CPI**

nCPI

PCS_PAGE_LPI

nLPI

PCS_PAGE_FACE_NAME

szFaceName

PCS_PAGE_MPL

nMPL

PCS_PAGE_MPP

nMPP

nCPI

The number of characters printed per inch.

To select Font CPI, set the HIWORD of nCPI to 1. LOWORD of nCPI will be ignored.

To select a particular CPI value, do the following:

1. Set the HIWORD of nCPI to 0.
2. Set the LOWORD of nCPI to the actual CPI value.

nLPI

The number of lines printed per inch.

To select Font LPI, set the HIWORD of nLPI to 1. LOWORD of nLPI will be ignored

To select a particular LPI value, do the following:

1. Set the HIWORD of nLPI to 0.
2. Set the LOWORD of nLPI to the actual LPI value.

szFaceName

Face name of the printer font. This must be a null-terminated string.

nMPL

Maximum number of lines that can be printed per page.

This is also called MPL (Maximum Print Lines). Supported range is 1 to 255.

nMPP

Maximum number of characters that can be printed per line.

This is also called MPP (Maximum Print Position). Supported range is 1 to 255.

ULONG * const pErrorInfo

Contains the extended error info when the API fails with the return code of PCS_FAILURE. If the detailed error information is not needed, this flag must be set to NULL by the caller.

This is a combination of bit flags that describe which members of the PAGEINFO structure could not be set successfully. The flags that are defined in PCSAPI32.H are as follows:

Flag

Valid members in the structure

PCS_PAGE_CPI

Only nCPI is not valid.

PCS_PAGE_LPI

Only nLPI is not valid.

PCS_PAGE_FACE_NAME

Only szFaceName is not valid.

PCS_PAGE_MPL

Only nMPL is not valid.

PCS_PAGE_MPP

Only nMPP is not valid.

Return Code

Return Code	Value	Meaning
PCS_SUCCESSFUL	0	Function ended successfully.
PCS_INVALID_ID	1	Incorrect session ID was specified.
PCS_INVALID_SESS_TYPE	2	Not supported for the host session type.
PCS_DIALOG_IN_USE	3	Failed because the host session Page Setup or Printer Setup dialog was in use.
PCS_PRINTING	4	Page settings cannot be changed because host session was printing.
PCS_PDT_MODE	5	Page settings cannot be changed because host session is in PDT mode.
PCS_FAILURE	6	Host session page settings are not fully applied. This could be because invalid data was given for some or all fields in the PAGEINFO structure. Examine pErrorInfo for details about settings that are not applied.

Return Code	Value	Meaning
PCS_SYSTEM_ERROR	9	A system error occurred.

Example

```

{
    ULONG Rc = 0, Error = 0;
    PAGEINFO *PageInfo;

    PageInfo = (PAGEINFO *) malloc(sizeof(PAGEINFO));
    memset(PageInfo, 0, sizeof(PAGEINFO));

    PageInfo->nFlags = PCS_PAGE_CPI | PCS_PAGE_LPI |
                     PCS_PAGE_FACE_NAME | PCS_PAGE_MPL |
                     PCS_PAGE_MPP;
    PageInfo->nCPI = MAKELONG(10, 0);
    PageInfo->nLPI = MAKELONG(8, 0);
    PageInfo->nMPL = 40;
    PageInfo->nMPP = 60;
    strcpy(PageInfo->szFaceName, "CourierPS");

    Rc = pcsSetPageSettings('A', PageInfo, &Error);

    if (Rc != PCS_SUCCESSFUL) {
        printf("Failure. Return code = %d\n", Rc);
        printf("Following members could not be set : ");

        if (Rc == PCS_FAILURE) {
            if (Error & PCS_PAGE_CPI) printf(" nCPI");
            if (Error & PCS_PAGE_LPI) printf(" nLPI");
            if (Error & PCS_PAGE_FACE_NAME) printf(" szFaceName");
            if (Error & PCS_PAGE_MPL) printf(" nMPL");
            if (Error & PCS_PAGE_MPP) printf(" nMPP");
            printf("\n");
        }
    }
    free(PageInfo);
}

```

Printer Setup Functions

The PCSAPI functions listed in this section enable you to control and retrieve the Z and I Emulator for Windows emulator session **Printer Setup** settings.

Restrictions

If the following restrictions are not met, the API will fail. The return code indicates the reason for the failure.

- The host session should not be printing when the API is invoked.
- The **File → Printer Setup** dialog should not be in use.

pcsGetPrinterSettings

3270	5250	VT
Yes	Yes	Yes

The **pcsGetPrinterSettings** function retrieves the host session printer settings (similar to the **File → Printer Setup** dialog settings).

Function Type

ULONG WINAPI pcsGetPrinterSettings(*char cShortSessionID*, *PRINTINFO * const pPrintInfo*, *ULONG * const pErrorInfo*)

Parameter Type and Description

char cShortSessionID

Presentation space short session ID.

PRINTINFO * const pPrintInfo

Pointer to PRINTINFO structure, where the printer settings are specified.

nFlags

Must be set to 0. This is ignored.

nBufSize

Size of the buffer allocated for the following fields:

- lpPDTFile
- lpPrtToDskAppFile
- lpPrtToDskSepFile
- lpPrinterName

If more than one of these members is retrieved in a single API call, then the caller must allocate the same size for all the buffers and pass that size in this member.

If this member is set to 0, the fields are ignored. The maximum size required for the buffers of the fields is returned in nSizeNeeded.

nSizeNeeded

The value of this member is determined by conditions related to the following fields:

- lpPDTFile
- lpPrtToDskAppFile
- lpPrtToDskSepFile
- lpPrinterName

The conditions are as follows:

- The value is the number of bytes needed, if the size of the buffer allocated by the caller is not big enough to return the fields listed above.
- The value is the maximum size of the required buffer, if more than one of the fields listed above are obtained by the caller.
- If nBufSize is set to 0 by the caller, this member contains the maximum size required for the buffers of the fields listed above.

bPromptDialog

Possible values are as follows:

- If TRUE, the Printer Setup dialog is shown before printing.
- If FALSE, the Printer Setup dialog is not shown before printing.

bPDTMode

Possible values are as follows:

- If TRUE, the host session is in PDT mode.
- If FALSE, the host session is in non-PDT mode (GDI mode).

lpPDTFile

Must be set to NULL if the caller is not interested in getting this member. The PDT file is returned if this is not a null pointer. This must point to the buffer of size nBufSize allocated by the caller.

When the API returns, this member contains one of the following:

- The fully qualified path name of the session PDT file.
- An empty string ("") if no PDT file is configured in the session.
- A truncated file name if the buffer size is not sufficient. The member nSizeNeeded contains the size of the buffer needed.

nPrtMode

This is an enumerated value that indicates the PrintMode of the connection. The enum data type PRINTMODE is defined in PCSAPI32.H. The nPrtMode setting must be one of the following:

- **PrtToDskAppend (Print to Disk-Append mode)**

This is equivalent to selecting the **Append** option in the host session **Printer Setup** → **Printer** → **Print to Disk** dialog.

- **PrtToDskSeparate (Print to Disk-Separate mode)**

This is equivalent to selecting the **Separate** option in the host session **Printer Setup → Printer → Print to Disk** dialog.

- **WinDefaultPrinter** (**Windows Default Printer** mode)

This is equivalent to selecting the **Use Windows Default Printer** option in the host session **Printer Setup** dialog.

- **SpecificPrinter** (**Specific Printer** mode)

This is equivalent to selecting a printer in the host session **Printer Setup** dialog, while leaving **Use Windows Default Printer** unchecked.

lpPrtToDskAppFile

Must be set to NULL if the caller is not interested in getting this member. The **Print to Disk-Append** file is returned if this is not a null pointer. This must point to the buffer of size `nBufSize` allocated by the caller.

When the API returns, this member contains one of the following:

- The fully qualified path name of the session **Print to Disk-Append** file.
- An empty string ("") if no **Print to Disk-Append** file is configured for the session.
- A truncated file name if the buffer size is not sufficient. The `nSizeNeeded` member contains the size of the buffer needed.

lpPrtToDskSepFile

Must be set to NULL if the caller is not interested in getting this member. The **Print to Disk-Separate** file is returned if this is not a null pointer. This must point to the buffer of size `nBufSize` allocated by the caller.

When the API returns, this member contains one of the following:

- The fully qualified path name of the session **Print to Disk-Separate** file.
- An empty string ("") if no **Print to Disk-Separate** file is configured for the session.
- A truncated file name if the buffer size is not sufficient. The `nSizeNeeded` member contains the size of the buffer needed.

lpPrinterName

Must be set to NULL if the caller is not interested in getting this member. The name of the printer is returned if this is not a null pointer. This must point to the buffer of size `nBufSize` allocated by the caller.

When the API returns, this member has one of the following:

- The name of the specific printer configured in the session, if the host session nPrtMode is SpecificPrinter.
- The name of the Windows default printer configured in the session, if the host session nPrtMode is WinDefaultPrinter.
- An empty string (""), if the host session nPrtMode is PrtToDskAppend or PrtToDskSeparate.
- A truncated printer name, if the buffer size is not sufficient. nSizeNeeded has the size of the buffer needed.

PrinterName must have the following format:

```
<Printer name> on <Port Name>
```

For example:

- IBM InfoPrint 40 PS on Network Port
- HP LaserJet 4050 Series PCL 6 on LPT1

ULONG * const pErrorInfo

This is filled with the extended error info when the API fails with the return code of PCS_FAILURE. pErrorInfo must be set to NULL by the caller, if the details of errors are not needed.

The following section describes the flags that are defined in PCSAPI32.H.

Flags for the pErrorInfo member of the PRINTINFO structure

PCS_PRINT_PRINTMODE_ERROR

PrintMode is not configured in the host session.

PCS_PRINT_PDTFILE_SIZEERR

The buffer size is not sufficient for lpPDTFile, so the file name is truncated. The nSizeNeeded member contains the actual size of the buffer required to return the PDT file.

PCS_PRINT_DSKAPPPFILE_SIZEERR

The buffer size is not sufficient for lpPrtToDskAppFile, so the file name is truncated. The nSizeNeeded member contains the actual size of the buffer required to return the **Print to Disk-Append** file.

PCS_PRINT_DSKSEPFIL_ SIZEERR

The buffer size is not sufficient for lpPrtToDskSepFile, so the file name is truncated. The nSizeNeeded member contains the actual size of the buffer required to return the **Print to Disk-Separate** file.

PCS_PRINT_PRINTERNAME_SIZEERR

The buffer size is not sufficient for lpPrinterName, so the printer name is truncated. The nSizeNeeded member contains the actual size of the buffer required to return the printer name.

Return Code

Return Code	Value	Meaning
PCS_SUCCESSFUL	0	The function ended successfully.
PCS_INVALID_ID	1	An incorrect session ID was specified.
PCS_DIALOG_IN_USE	3	Failed because the host session Page Setup or Printer Setup dialog was in use.
PCS_PRINTING	4	The printer settings could not be changed because the host session was printing. The application must retry later
PCS_FAILURE	6	Some printer settings could not be retrieved successfully. p-ErrorInfo contains detailed error information on which settings could not be retrieved.
PCS_SYSTEM_ERROR	9	A system error occurred.

Example

```
{
    ULONG Rc = 0, Error=0, Size;
    PRINTINFO *PrintInfo;

    PrintInfo = (PRINTINFO *) malloc(sizeof(PRINTINFO));
    memset(PrintInfo, 0, sizeof(PRINTINFO));

    PrintInfo->nBufSize = 0;

    Rc = pcsGetPrinterSettings('A', PrintInfo, &Error);
    if (Rc != PCS_SUCCESSFUL)
        printf("Failure. Return code = %d\n", Rc);
    else {
        Size = PrintInfo->nSizeNeeded;
        PrintInfo->nBufSize = Size;
        PrintInfo->lpPDFile = (char *)malloc(sizeof(char) * Size);
        PrintInfo->lpPrtToDskAppFile = (char *)malloc(sizeof(char) * Size);
        PrintInfo->lpPrtToDskSepFile = (char *)malloc(sizeof(char) * Size);
        PrintInfo->lpPrinterName = (char *)malloc(sizeof(char) * Size);
        Rc = pcsGetPrinterSettings('A', PrintInfo, &Error);

        if (Rc != PCS_SUCCESSFUL)
            printf("Failure. Return code = %d, Extended Error = 0x%08x\n", Rc, Error);
        else {
            if (PrintInfo->bPromptDialog)
                printf("PromptDialog\n");
            else
                printf("No PromptDialog\n");
            if (PrintInfo->bPDTMode)
                printf("PDT Mode\n");
            else
                printf("Not PDT Mode\n");

            switch(PrintInfo->nPrtMode) {
```

```

    case PrtToDskAppend:
        printf("Print to Disk-Append Mode\n");
        break;
    case PrtToDskSeparate:
        printf("Print to Disk-Separate Mode\n");
        break;
    case SpecificPrinter:
        printf("Specific Printer Mode\n");
        break;
    case WinDefaultPrinter:
        printf("Windows Default Printer Mode\n");
        break;
}
if (PrintInfo->lpPDTFile[0] == '\\0')
    printf("No PDT File configured\n");
else
    printf("PDT File = %s\n", PrintInfo->lpPDTFile);
if (PrintInfo->lpPrtToDskAppFile[0] == '\\0')
    printf("No Disk Append File configured\n");
else
    printf("DiskAppend File=%s\n", PrintInfo->lpPrtToDskAppFile);
if (PrintInfo->lpPrtToDskSepFile[0] == '\\0')
    printf("No Disk Separate File configured\n");
else
    printf("DiskSeparate File=%s\n", PrintInfo->lpPrtToDskSepFile);
if ((PrintInfo->nPrtMode == SpecificPrinter) ||
    (PrintInfo->nPrtMode == WinDefaultPrinter))
    printf("Printer = %s\n", PrintInfo->lpPrinterName);
}
free(PrintInfo->lpPDTFile);
free(PrintInfo->lpPrtToDskAppFile);
free(PrintInfo->lpPrtToDskSepFile);
free(PrintInfo->lpPrinterName);
}
free(PrintInfo);
}

```

pcsSetPrinterSettings

3270	5250	VT
Yes	Yes	Yes

The **pcsSetPrinterSettings** function controls the host session printer settings (similar to the **File → Printer Setup** dialog settings).

Function Type

ULONG WINAPI pcsSetPrinterSettings(*char cShortSessionID, const PRINTINFO * const pPrintInfo, ULONG * const pErrorInfo*)

Parameter Type and Description

char cShortSessionID

Presentation space short session ID.

const PRINTINFO * const pPrintInfo

Pointer to PRINTINFO structure, where the printer settings are mentioned.

nFlags

Combination of bit flags that indicates which members in the structure are valid. These flags can be used independently or by ORing them together to restore the property page (defined in PCSAPI32.H). The flags, along with the corresponding valid members in the structure, are as follows:

Flag**Valid members in the structure****PCS_PRINT_PDT**

bPDTMode, lpPDTFile

PCS_PRINT_PRINTMODE

nPrtMode, lpPrtToDskAppFile, lpPrtToDskSepFile, lpPrinterName

PCS_PRINT_PROMPT_DIALOG

bPromptDialog

nBufSize

Must be set to 0. This is ignored.

nSizeNeeded

Must be set to 0. This is ignored.

bPromptDialog

Possible values are as follows:

- If TRUE, the Printer Setup dialog is shown before printing.
- If FALSE, the Printer Setup dialog is not shown before printing.

bPDTMode

Possible values are as follows:

- If TRUE, the connection is set to PDT mode.
- If FALSE, the connection is set to non-PDT mode (GDI mode).

lpPDTFile

Used only if bPDTMode is set to TRUE. This is ignored if bPDTMode is set to FALSE.

This is a null-terminated string containing the name of the PDT file and must be one of the following:

- NULL

The PDT file that is currently configured in the connection is used. If there is no PDT file already configured in the connection, the API fails with an exception.

- File name, without the path

lpPDTFile in the PDFPDT subfolder in the Z and I Emulator for Windows installation path is used.

- Fully qualified path name of the file

If lpPDTFile does not exist, the API fails.

nPrtMode

This is an enumerated value that indicates the PrintMode of the connection. The enum data type PRINTMODE is defined in PCSAPI32.H. The nPrtMode setting must be one of the following:

- **PrtToDskAppend (Print to Disk-Append mode)**

This is equivalent to selecting the **Append** option in the host session **Printer Setup** → **Printer** → **Print to Disk** dialog.

- **PrtToDskSeparate (Print to Disk-Separate mode)**

This is equivalent to selecting the **Separate** option in the host session **Printer Setup** → **Printer** → **Print to Disk** dialog.

- **WinDefaultPrinter (Windows Default Printer mode)**

This is equivalent to selecting the **Use Windows Default Printer** option in the host session **Printer Setup** dialog.

- **SpecificPrinter (Specific Printer mode)**

This is equivalent to selecting a printer in the host session **Printer Setup** dialog, while leaving the **Use Windows Default Printer** option unchecked.

lpPrtToDskAppFile

This is used only if nPrtMode is set to PrtToDskAppend.

This is a null-terminated string containing the name of the **Print to Disk-Append** file and must be one of the following:

- NULL

The file that is currently configured for the **PrtToDskAppend** mode in the connection is used. If there is no PDT file already configured in the connection, the API will fail.

- File name, without the path

The user-class application data directory path is used to locate the file. If the file exists, it is used. Otherwise, it will be created when printing is complete.

- Fully qualified path name of the file

The directory must exist in the path, or the API will fail. It is not necessary that the file exist in the path.

IpPrtToDskSepFile

The possible values are as follows:

- Fully qualified path name of the **Print to Disk-Separate** file for the session.
- An empty string ("") if no **Print to Disk-Separate** file is configured for the session.
- A truncated file name if the buffer size is not sufficient. The `nSizeNeeded` member contains the size of the buffer needed.

IpPrinterName

This is used only if `nPrtMode` is set to `SpecificPrinter`. It is ignored otherwise. This is a null-terminated string containing the printer name. If the printer does not exist, this member fails.

`PrinterName` must have the following format:

```
<Printer name> on <Port Name>
```

For example:

- IBM InfoPrint 40 PS on Network Port
- HP LaserJet 4050 Series PCL 6 on LPT1

ULONG * const pErrorInfo

This is filled with the extended error info when the API fails with the return code of `PCS_FAILURE`. `pErrorInfo` must be set to NULL by the caller, if the details of errors are not needed.

The following section describes the flags that are defined in `PCSAPI32.H`.

Flags for the pErrorInfo member of the PRINTINFO structure

PCS_PRINT_PDTMODE_ERROR

This can occur for one of one of the following reasons:

- bPDTMode is set to TRUE, lpPDTFile is set to NULL, and there is no PDT file already configured for the host session.
- nPrtMode is set to PrtToDskAppend or PrtToDskSeparate, PCS_PRINT_PDT is not set in nFlags, and the host session is not already in PDT mode.
- nPrtMode is set to PrtToDskAppend or PrtToDskSeparate and bPDTMode is set to FALSE.

PCS_PRINT_PDTFILE_ERROR

The file or the path specified in lpPDTFile was not found.

PCS_PRINT_PRTTODSK_FILE_ERROR

This can occur for one of one of the following reasons:

- The folder specified in the field lpPrtToDskAppFile or lpPrtToDskSepFile does not exist or does not have write access.
- An extension is specified in the field lpPrtToDskSepFile.

PCS_PRINT_PRINTMODE_ERROR

nPrtMode cannot be set successfully. This can occur for one of the following reasons:

- The value of nPrtMode is not one of the enumerated constants of the PRINTMODE enum data type.
- nPrtMode is set to PrtToDskAppend, lpPrtToDskAppFile is set to NULL, and there is no **Print to Disk-Append** file already configured in the host session.
- nPrtMode is set to PrtToDskSeparate, lpPrtToDskSepFile is set to NULL, and there is no **Print to Disk-Separate** file already configured in the host session.
- nPrtMode is set to SpecificPrinter and the printer given in the lpPrinterName field was not found.
- nPrtMode is set to WinDefaultPrinter and there is no default Windows® printer configured in the system.
- bPDTMode is set to FALSE and PCS_PRINT_PRINTMODE is not set in nFlags, but the host session PrintMode is PrtToDskAppend or PrtToDskSeparate.

Return Code

Return Code	Value	Meaning
PCS_SUCCESSFUL	0	The function ended successfully.
PCS_INVALID_ID	1	An incorrect session ID was specified.

Return Code	Value	Meaning
PCS_DIALOG_IN_USE	3	Failed because the host session Page Setup or Printer Setup dialog was in use.
PCS_PRINTING	4	The printer settings could not be changed because the host session was printing. The application must retry later.
PCS_FAILURE	6	No host session printer settings were applied. This might occur because invalid data was given for some or all of the fields in the PRINTINFO structure. pErrorInfo contains details about the errors.
PCS_SYSTEM_ERROR	9	A system error occurred.

Example

```

{
    ULONG Rc = 0, Error=0;
    PRINTINFO *PrintInfo;
    char PDTFile[] = "epson.pdt";
    char SepFile[] = "DiskSep";

    PrintInfo = (PRINTINFO *) malloc(sizeof(PRINTINFO));
    memset(PrintInfo, 0, sizeof(PRINTINFO));

    PrintInfo->nFlags = PCS_PRINT_PDT | PCS_PRINT_PRINTMODE |
        PCS_PRINT_PROMPT_DIALOG;
    PrintInfo->nBufSize = 0;
    PrintInfo->nSizeNeeded = 0;
    PrintInfo->bPDTMode = TRUE;
    PrintInfo->lpPDTFile =
        (char *)malloc(sizeof(char) * (strlen(PDTFile)+1));
    strcpy(PrintInfo->lpPDTFile, PDTFile);
    PrintInfo->nPrtMode = PrtToDskSeparate;
    PrintInfo->lpPrtToDskSepFile =
        (char *)malloc(sizeof(char) * (strlen(SepFile)+1));
    strcpy(PrintInfo->lpPrtToDskSepFile, SepFile);
    PrintInfo->bPromptDialog = TRUE;
    Rc = pcsSetPrinterSettings('A', PrintInfo, &Error);
    if (Rc != PCS_SUCCESSFUL)
        printf("Failure. Return code = %d, Extended Error = 0x%08x\n", Rc, Error);
    free(PrintInfo->lpPDTFile);
    free(PrintInfo->lpPrtToDskSepFile);
    free(PrintInfo);
}

```


Chapter 6. Troubleshooting for Emulator programming

You can use the following self-help information resources and tools to help you troubleshoot problems:

- Refer to the release information for your product for known issues, workaround, and troubleshooting information.
 - Check if a download or fix is available to resolve your problem.
 - Search the available knowledge bases to see if the resolution to your problem is already documented.
 - If you still need help, contact HCL Software Support and report your problem.
-

Partial EHLLAPI input on Z and I Emulator for Windows host screen

Problem

Truncated command text was sent to a host when using HCL Z and I Emulator for Windows.

Cause

If an EHLLAPI application sends a SYSREQ key to the host and then tries to input a command onto the host screen, sometimes only a truncated part of the command is sent to the host. This problem occurs due to lack of synchronization between the SYSREQ processing at the Z and I Emulator for Windows host side and the input of commands from the EHLLAPI application.

When the application sends a SYSREQ command to the host, the following situations occur:

- The OIA is updated to indicate that you are in a SSCP-LU session.
- The Z and I Emulator for Windows session sends the AO command (the SYSREQ) to the 3270 host.

As soon as the host receives the SYSREQ, it responds to Z and I Emulator for Windows with the 0x15 or NL (NewLine) code. When Z and I Emulator for Windows processes this NL command by filling the rest of the line with NULLs, and moving the cursor to the beginning of the next line.

A problem occurs when the EHLLAPI application continues to input various commands in the host screen (through the SendKeys function), even before the Z and I Emulator for Windows session has received the NL command from the host and processed it. As a result, a part of the input command is first entered onto the screen, while the NL command is processed and the cursor is moved over to the next line. Then the remaining part of the command is input on the next line. Thus, only the truncated second part of the command is sent to the host, causing erroneous results.

Resolution

The solution for this problem is to force the EHLLAPI application to wait until the NL command is received and processed, before continuing to input the commands to the host screen. Once the session has notified the EHLLAPI application that the host response for SYSREQ has been processed, the

EHLAPI application can then continue with its input (because the session is now in the right state to accept new input). To accomplish this, use the following EHLAPI function calls:

```
Start_Host_Notification (23)
Pause (18)
Set_Session_Parameters (9)
Query_Host_Update (24).
```

Possible code in the EHLAPI application is as follows:

- Call Sendkeys(@A@H). This sends the SYSREQ command to the session.
- Call StartHostNotify with input B, where B indicates notification of both OIA and PS. This tells the session to notify the EHLAPI application when the session's OIA and/or PS is updated by the host.
- Call Pause, specifying a sufficient timeout period. This causes the EHLAPI application to wait until the session notifies it of a host update to the session's OIA and/or PS. This occurs when the session receives the most-awaited host response for the SYSREQ command. Note that if the timeout value has been exceeded, and no host notification has been received, the Pause function call still returns.

Also, for this Pause call to work, you must use the Set_Session_Parameters (9) function call to enable the IPAUSE option. This is required because it tells the Pause API call to return when the host notifies the session of an OIA and/or PS update.

If Pause has returned due to an OIA/PS update (host notification), it has a return value of 26. If this is the case, you are ready to send the host command. Otherwise, you must wait again for the host response.

The EHLAPI application can continue with the command once it knows that either the OIA or the Presentation Space (or both) has been updated by the host. The QueryHostUpdate is used to check what was updated: that is, whether the OIA alone was updated (return code 21), or the PS alone was updated (return code 22) or whether both the OIA and the PS were updated (return code 23).

For example, the EHLAPI code might resemble the following part:

```
Send Keys(@A@H) /* Send SYSREQ command to the host */

Start Host Notification with 'B' in byte 2 /* Enable notification to EHLAPI application
                                           when session's OIA and/or PS are updated */

Set Session Parms with IPAUSE option /* Allow Pause to be interrupted */

Label WW:

Pause for 15 seconds /* 15 secs is a sample time-out value */

retVal = Query Host Update /* Store return value of QueryHostUpdate() into retVal */

If (retVal = 21 or 22 or 23) /* OIA and/or PS was updated */
```

```

Send Keys("Your Input Command to host") /* Send input command to host */

else

goto (Label WW)

Stop Host Notification /* Disable host notification */

```

This is the most appropriate solution for this problem, because the EHLLAPI application waits for the exact minimum time required to allow the session to receive and process the SYSREQ host response, before sending its command input.

Another solution is to add a delay [for example, Sleep(1000)] in the EHLLAPI application between the SYSREQ command and the subsequent command, so that the session has enough time to receive and process the host response. However, this solution is not the best, because the delay might be too little or might be excessive.

Refer to RFC 2355 (TN3270 Enhancements) for more information about the 3270 SYSREQ functionality.

HCL Z and I Emulator for Windows VBHLLAPI sample does not run in FDCC Windows Vista

Problem

The HCL Z and I Emulator for Windows VBHLLAPI sample uses controls provided by comdlg32.ocx, which is not installed in the Federal Desktop Core Configuration (FDCC) of Microsoft Windows Vista.

Cause

VBHLLAPI uses ActiveX and Common Dialog controls that are provided by the Microsoft comdlg32.ocx module. For security purposes, the FDCC of Windows Vista does not contain this particular module.

Resolution

The FDCC version of Windows Vista is customized, and changes are not recommended.

If HLLAPI samples containing VBHLLAPI need to be run, then the comdlg32.ocx module must be copied from a standard Windows Vista machine into the `\Windows\System32\` directory of the FDCC Windows Vista installation.

Then reboot the system for the change to take effect.

Appendix A. Query Reply Data Structures Supported by EHLLAPI

This appendix lists and defines the query reply structures supported by the EHLLAPI structured field interface for PC/3270. Refer to *IBM 3270 Information Display System Data Stream Programmer's Reference* or, in the case of an IBM licensed program, the documentation for the specific licensed program.



Note:

1. EHLLAPI must scan the query reply buffers to locate the destination/origin ID (DOID) self-defining parameter (SDP) for the structured field support to work and be reliable. The DOID field is then filled in with the assigned ID.
2. The application should build the query reply data structures in the application's private memory.
3. Only cursory checking is performed on the query reply data. Only the ID and the length of the structure are checked for validity.
4. The 2-byte length field at the beginning of each query reply **is not byte reversed**.
5. Only one distributed data management (DDM) base-type connection is allowed per host session. If the DDM connection supports the SDP for the DOID, multiple connections are allowed.
6. If a nonzero return code is received indicating that an application is already connected to the selected session (RC 32 or 39), use that presentation space with caution. Conflicts with File Transfer, and other EHLLAPI applications might result.

The DDM Query Reply

Several DDM query reply formats are supported. Here are some of them:

Table 15. DDM Query Reply Base Format

Offset	Length	Content	Meaning
0	1 word	Length	Length of structure
2	1 byte	X'81'	Query reply ID
3	1 byte	X'95'	Query reply type
4-5	2 bytes	FLAGS	Reserved
6-7	2 bytes	LIMIN	Maximum DDM bytes allowed in inbound transmission
8-9	2 bytes	LIMOUT	Maximum DDM bytes allowed in outbound transmission
10	1 byte	NSS	Number of subsets identifier
11	1 byte	DDMSS	DDM subset identifier

DDM Application Name Self-Defining Parameter

The DDM application name self-defining parameter provides the host application with the name of the application containing control of the DDM auxiliary device. The controlling application is identified by the DOID in the Direct Access self-defining parameter.

This self-defining parameter is optional, but it is necessary if a host application is to identify a distinct DDM auxiliary device when more than one application is in existence at a remote workstation.

Table 16. DDM Application Name Self-Defining Parameter

Offset	Length	Content	Meaning
0	1 byte	Length	Parameter length
1	1 byte	X'02'	DDM application name
2–n	n-2 bytes	NAME	Name of the remote application program

NAME

The name consists of 8 characters or less and is the means by which a host application can relate to an application in a remote workstation. It is the responsibility of the host and remote application users to ensure that the name is understood by the application at each end.

PCLK Protocol Controls Self-Defining Parameter

The PCLK Protocol Controls self-defining parameter indicates that the PCLK Protocol Controls structured field, ID = X'1013', can be used for both inbound and outbound in data streams destined to or from the DDM auxiliary device processor.

Table 17. DDM PCLK Auxiliary Device Self-Defining Parameter

Offset	Length	Content	Meaning
0	1 byte	X'04'	Parameter length
1	1 byte	X'03'	PCLK protocol controls
2–3	2 bytes	VERS	Protocol version

VERS

The value given in VERS is used to indicate the versions of PCLK installed in the terminal at the time the query reply is returned. For example, X'0001' indicates PCLK Version 1.1.

Refer to *IBM 3270 Information Display System Data Stream Programmer's Reference* for the field definitions for this query reply.

Base DDM Query Reply Formats

The following query reply formats are *examples* of some of the Base + SDP (self-defining parameter) combinations possible. Not all of the combinations are shown.

Table 18. Base DDM Query Reply Format with Name and Direct Access Self-Defining Parameters

Offset	Length	Content	Meaning
0	1 word	Length	Length of structure (includes self-defining parameters)
2	1 byte	X'81'	Query reply ID
3	1 byte	X'95'	Query Reply type
4–5	2 bytes	FLAGS	Reserved
6–7	2 bytes	LIMIN	Maximum DDM bytes allowed in inbound transmission
8–9	2 bytes	LIMOUT	Maximum DDM bytes allowed in outbound transmission
10	1 byte	NSS	Number of subsets supported
11	1 byte	DDMSS	DDM subset identifier
12	1 byte	Length (n+2)	Parameter length
13	1 byte	X'02'	DDM application name
14– (13+n)	n bytes	Name	Name of the remote application program
14+n	1 byte	X'04'	Parameter length
15+n	1 byte	X'01'	Direct access ID
16+n – 17+n	2 bytes	DOID	Destination/origin ID assigned by the subsystem

The self-defining parameters begin at offsets 12 and $(14 + n)$ where n is the length of the application name supplied at offset 14.

Refer to *IBM 3270 Information Display System Data Stream Programmer's Reference* for the field definitions for this query reply.

Table 19. Base DDM Query Reply Format with Direct Access and Name Self-Defining Parameters

Offset	Length	Content	Meaning
0	1 word	Length	Length of structure (includes self-defining parameters)
2	1 byte	X'81'	Query reply ID

Table 19. Base DDM Query Reply Format with Direct Access and Name Self-Defining Parameters (continued)

Offset	Length	Content	Meaning
3	1 byte	X'95'	Query reply type
4–5	2 bytes	FLAGS	Reserved
6–7	2 bytes	LIMIN	Maximum DDM bytes allowed in inbound transmission
8–9	2 bytes	LIMOUT	Maximum DDM bytes allowed in outbound transmission
10	1 byte	NSS	Number of subsets supported
11	1 byte	DDMSS	DDM subset identifier
12	1 byte	X'04'	Parameter length
13	1 byte	X'01'	Direct access ID
14–15	2 bytes	DOID	Destination/origin ID assigned by the subsystem
16	1 byte	Length (n+2)	Parameter length
17	1 byte	X'02'	DDM application name
16+n – 17+n	n bytes	Name	Name of the remote application program

The self-defining parameters begin at offsets 12 and 16.

Refer to *IBM 3270 Information Display System Data Stream Programmer's Reference* for the field definitions for this query reply.

The IBM Auxiliary Device Query Reply

The Auxiliary Device Query Reply is used to indicate to the host application the support of an IBM auxiliary device that uses a data stream defined by IBM, refer to *IBM 3270 Information Display System Data Stream Programmer's Reference* for more details.

When the function is supported, the query reply is transmitted inbound in reply to a Read Partition structured field specifying Query or Query List (QCODE List = X'9E', Equivalent, or All).

When a workstation supports multiple auxiliary devices, the IBM auxiliary devices query reply must be sent for each device.

Optional Parameters

All parameters shown in the base part of the query reply must be present. Parameters not used are set to X'00'.

At least one self-defining parameter must be present.

Table 20. IBM Auxiliary Device Base Format with Direct Access Self-Defining Parameter

Offset	Length	Content	Meaning
0–1	1 word	Length	Length of structure (includes self-defining parameters)
2	1 byte	X'81'	Query reply ID
3	1 byte	X'9E'	IBM auxiliary device reply
4	1 byte	FLAGS	Reserved
	BIT 0	QUERY B'1'	Read Part (Query, Query List) Auxiliary device supports Query
	1–7	RES	Reserved, must be B'0's
5	1 byte	FLAGS	Reserved
6–7	2 bytes	LIMIN	Maximum DDM bytes allowed in inbound transmission
8–9	2 bytes	LIMOUT	Maximum DDM bytes allowed in outbound transmission
10	1 byte	TYPE X'01' X'02' Others	Type of auxiliary device supported IBM auxiliary device display IBM auxiliary device printer Reserved
11	1 byte	X'04'	Parameter length
12	1 byte	X'01'	Direct access
13–14	1 word	DOID	Destination/origin ID assigned by the subsystem

QUERY	<p>This bit must be set to B'1' for all IBM auxiliary devices to indicate that it supports receiving a Read Partition (Query, Query List). The host applications can then use a Read Partition directed to the auxiliary device to determine its characteristics. The destination/origin structured field is used to direct the Read Partition structured field to the auxiliary device.</p> <p>The minimum support level for the IBM auxiliary device is to return the Null query reply in response to the Read Partition.</p>
LIMIN	States the maximum number of bytes that can be sent in an inbound transmission. A LIMIN value of X'0000' indicates no implementation limit on the number of bytes transmitted inbound.
LIMOUT	States the maximum number of bytes that can be sent to an IBM auxiliary device in an outbound transmission. A LIMOUT value of X'0000' indicates no implementation limit on the number of bytes transmitted outbound.
TYPE	Identifies the auxiliary device being supported. Two values are valid. One identifies an auxiliary display and the other identifies an auxiliary printer. All other values are reserved.

The IBM auxiliary device processor supports two self-defining parameters, 01 and 03. These are defined in [Table 21: IBM Auxiliary Device Direct Access Self-Defining Parameter on page 225](#).

Direct Access Self-Defining Parameter

The direct access self-defining parameter provides the ID for use in the destination/origin structured field in the direct access of the IBM auxiliary device.

This SDP is always required to accompany the base query reply.

Table 21. IBM Auxiliary Device Direct Access Self-Defining Parameter

Offset	Length	Content	Meaning
0	1 byte	X'04'	Parameter length
1	1 byte	X'01'	Direct access ID
2–3	2 bytes	DOID	Destination/origin ID

DOID

The value in these bytes is used in the ID field of the destination/origin structured field to identify the auxiliary device as the destination or origin of the data that follows.

PCLK Protocol Controls Self-Defining Parameter

The presence of the PCLK protocol controls self-defining parameter indicates that the PCLK protocol controls structured field, ID = X'1013', can be used for both inbound and outbound in data streams destined to or from the IBM auxiliary device processor.

Table 22. IBM Auxiliary Device PCLK Self-Defining Parameter

Offset	Length	Content	Meaning
0	1 byte	X'04'	Parameter length
1	1 byte	X'03'	PCLK protocol controls
2–3	2 bytes	VERS	Protocol version

VERS

The value given in VERS is used to indicate the versions of PCLK installed in the terminal at the time the query reply is returned. For example, X'0001' indicates PCLK version 1.1.

Refer to *IBM 3270 Information Display System Data Stream Programmer's Reference* for the field definitions for this query reply.

The Product-Defined Query Reply

This query reply is used by IBM products using registered subidentifiers within the X'9C' data structure. The Product-Defined Data Stream query reply indicates support of a 3270DS workstation auxiliary device that uses an IBM

product-defined data stream. The data stream is *not* defined by a format architecture document having an identifiable control point such as an architecture review board.

When an auxiliary device supports an IBM product-defined data stream, this query reply is transmitted inbound in reply to a Query List (QCODE List = X'9C' or All).

Optional Parameters

All parameters shown in the base part of the query reply and the direct access self-defining parameter must be present.

The format of the Product-Defined query reply is as follows:

Table 23. IBM Product-Defined Query Reply Base Format

Offset	Length	Content	Meaning
0–1	1 word	Length	Length of structure (includes self-defining parameters)
2	1 byte	X'81'	Query reply ID
3	1 byte	X'9C'	IBM product-defined data stream
4–5	2 bytes	FLAGS	Reserved
6	1 byte	REFID	Reference identifier
7	1 byte	SSID	Subset identifier
8	1 byte	X'04'	Parameter length
9	1 byte	X'01'	Direct access
10–11	1 word	DOID	Destination/origin ID assigned by the subsystem

Valid values for REFID (offset 6) and SSID (offset 7) of the Product-Defined query reply are as follows:

Table 24. Valid REFID and SSID Values for the IBM Product-Defined Query Reply

REFID	SSID	Product and Data Stream Documentation
X'01'		5080 Graphics System: This reference ID indicates the 5080 Graphics System data stream is supported by the auxiliary device. Descriptions of the 5080 Graphics Architecture, structured field, subset ID, DOID, and associated function sets are defined in <i>IBM 5080 Graphics System Principles of Operation</i>
	X'01' X'02'	5080 HGFD Graphics Subset 5080 RS232 Ports Subset
X'02'		WHIP API (replaced by SRL name when written)

Table 24. Valid REFID and SSID Values for the IBM Product-Defined Query Reply (continued)

REFID	SSID	Product and Data Stream Documentation
		This reference ID indicates that the WHIP API data stream is supported by the auxiliary device. A description of the WHIP API architecture is defined in <i>IBM RT PC Workstation Host Interface Program Version 1.1 User's Guide and Reference Manual</i>
	X'01'	WHIP Subset 1
X'03' to X'FF'		All other values are reserved.

The IBM product-defined processor supports only the direct access self-defining parameter. It is defined in [Table 25: IBM Product-Defined Direct Access Self-Defining Parameter on page 227](#).

Direct Access Self-Defining Parameter

The presence of the Direct Access ID self-defining parameter indicates that the auxiliary device can be accessed directly by using the destination/origin structured field. When multiple auxiliary devices are supported that use a product-defined data stream, separate Product-Defined Data Stream query replies must be provided, each of which has a unique DOID.

Table 25. IBM Product-Defined Direct Access Self-Defining Parameter

Offset	Length	Content	Meaning
0	1 byte	X'04'	Parameter length
1	1 byte	X'01'	Direct access ID
2–3	2 bytes	DOID	Destination/origin ID

DOID

The value in these bytes is used in the ID field of the destination/origin structured field to identify the auxiliary device as the destination or origin of the data that follows.

The Document Interchange Architecture Query Reply

This query reply indicates the Document Interchange Architecture (DIA) function set supported. The format of the DIA Query Reply is as follows:

Table 26. IBM DIA Base Format

Offset	Length	Content	Meaning
0	1 word	Length	Length of structure (includes self-defining parameters)
2	1 byte	X'81'	Query reply ID
3	1 byte	X'97'	IBM DIA
4–5	2 bytes	FLAGS	Reserved

Table 26. IBM DIA Base Format (continued)

Offset	Length	Content	Meaning
6–7	2 bytes	LIMIN	Maximum DDM bytes allowed in inbound transmission
8–9	2 bytes	LIMOUT	Maximum DDM bytes allowed in outbound transmission
10	1 byte	NFS	Number of 3-byte function set IDs that follow
11–13	3 bytes	DIAFS	DIA function set identifier
14– (13+(N*3))	N*3 bytes	DIAFSS	Additional DIA function set IDs
14+(N*3)	1 byte	X'04'	Parameter length
15+(N*3)	1 byte	X'01'	Direct access
16+(N*3)	1 word	DOID	Destination/origin ID assigned by the subsystem

The DIA auxiliary device processor supports only the direct access self-defining parameter. It is defined in [Table 27: IBM Product-Defined Direct Access Self-Defining Parameter on page 228](#).

The presence of the direct access ID self-defining parameter indicates that the auxiliary device can be accessed directly by using the destination/origin structured field.

Table 27. IBM Product-Defined Direct Access Self-Defining Parameter

Offset	Length	Content	Meaning
0	1 byte	X'04'	Parameter length
1	1 byte	X'01'	Direct access ID
2–3	2 bytes	DOID	Destination/origin ID

DOID

The value in these bytes is used in the ID field of the destination/origin structured field to identify the auxiliary device as the destination or origin of the data that follows.

Refer to *IBM 3270 Information Display System Data Stream Programmer's Reference* for the field definitions for this query reply.

Appendix B. Differences from Communication Manager/2 EHLLAPI

This appendix describes the differences between EHLLAPI of Z and I Emulator for Windows and EHLLAPI for Communication Manager/2.

The following EHLLAPI functions are different from those with the same names in Communication Manager/2. You need to understand the differences when you use these functions:

- **Set Session Parameter** (9)
- **Copy OIA** (13)
- **Copy String to PS** (15)
- **Storage Manager** (17)
- **Copy String to Field** (33)
- **Get Key** (51)
- **Window Status** (104)
- **Query Sessions** (10)
- **Connect for Structured Field** (120)
- **Allocate Communications Buffer** (123)
- ASCII mnemonics

Set Session Parameter (9)

Set Options

Z and I Emulator for Windows does not provide the following set options provided by Communication Manager:

OLDOIA, NEWOIA
COMPCASE, COMPICASE
OLD5250OIA, NEW5250OIA

Return Parameters

When the **Set Session Parameter** (9) function is terminated, Communication Manager returns a length of the valid data string as the third parameter, the data string length. However, Z and I Emulator for Windows returns a number of the valid set options as the data string length.

EAB Option

In Communication Manager/2, a color remap affects the value of the character color in the EAB attribute copied by **Copy PS** (5) or **Copy PS to String** (8) function when the EAB option is specified in the **Set Session Parameter** (9) function.

In Z and I Emulator for Windows, however, the value of the character color in the EAB attribute depends on the contents of the presentation space regardless of a color remap, and it is not affected by a color remap.

Copy OIA (13)

The **Copy OIA** (13) function has the following differences between Communication Manager/2 and Z and I Emulator for Windows. For more information of the group and the column positions, refer to [Copy OIA \(13\) on page 49](#).

- Byte Position 21
 - Z and I Emulator for Windows returns X'F6'.
 - Communication Manager/2 returns X'20'.
- Byte Positions 61–63
 - Z and I Emulator for Windows does not return the printer information.
 - Communication Manager/2 returns the printer information.
- Group 3: Shift State

Communication Manager/2 does not return the value of bit 2. Bit 2 is reserved, and bit 0 contains both the Upper Shift and the Caps Lock.

- Group 8 Byte 1: Input Inhibited
 - Z and I Emulator for Windows does not return bit 6 (Device not working).
 - Communication Manager/2 can return bit 6.
- Group 8 Byte 3: Input Inhibited
 - Z and I Emulator for Windows does not return bit 1 (Operator unauthorized) and bit 2 (Operator unauthorized -f).
 - Communication Manager/2 can return bits 1 and 2.
- Group 8 Byte 4: Input Inhibited
 - Z and I Emulator for Windows does not return bit 2 (System wait).
 - Communication Manager/2 can return bit 2.
- Group 10: Highlight Group 2
 - Z and I Emulator for Windows does not return bit 0 (Selected).
 - Communication Manager/2 can return bit 0.
- Group 11: Color Group 2
 - Z and I Emulator for Windows does not return bit 0 (Selected).
 - Communication Manager/2 can return bit 0.
- Group 13: Printer Status
 - In Z and I Emulator for Windows, this group is reserved.
 - Communication Manager/2 can return this group.
- Group 14: Graphics

Communication Manager/2 does not return bit 0 (Graphic cursor).

Copy String to PS (15)

In Communication Manager/2, the EAB option of the **Set Session Parameter** (9) function affects the **Copy String to PS** function. When you specify the EAB option, pass the attribute data that has the same size as the text data to the function with the text data.

In Z and I Emulator for Windows, however, the data to be passed is only text data regardless of EAB option. If you want to use the same interface with Communication Manager/2, use the `PUTEAB` option of **Set Session Parameter** (9).

Storage Manager (17)

Storage Manager (17) function provided by Communication Manager/2 is not supported by Z and I Emulator for Windows. Use the APIs provided by Windows® to allocate the memory for the applications.

Copy String to Field (33)

In Communication Manager/2, when the EAB option of the **Set Session Parameter** (9) function is specified, the attribute data is passed to the function as a part of the data. Therefore, when you specify the EAB option, pass the attribute data that has the same size as the text data to the function with the text data.

In Z and I Emulator for Windows, however, the EAB option does not affect the data contents of the **Copy String to Field** (33) function. The data to be passed is not the attribute data, but only the text data. If you want to use the same interface with Communication Manager/2, use the `PUTEAB` option of **Set Session Parameter** (9).

Get Key (51)

Communication Manager/2 returns shift state using @A, @S, or @r, if the shift state of a passed key is not a key or function recognized by the emulator session. Z and I Emulator for Windows does not support these ASCII mnemonics.

Window Status (104)

EHLLAPI function 104 (PM_WINDOW_STATUS) 'query extended status' command (0x03) will return the handle of the emulator presentation space window. This is consistent with the definition of the function and the Communication Manager/2 implementation. However, Z and I Emulator for Windows EHLLAPI returns the handle of the frame window. EHLLAPI applications written for Z and I Emulator for Windows using this function need to use the parent of the window handle returned.

Query Sessions (10)

In Communication Manager/2, the descriptor for personal computer is returned. However, the descriptor is not returned in Z and I Emulator for Windows.

Connect for Structured Fields (120)

The event object for communication connection status provided by Communication Manager/2 is not in Z and I Emulator for Windows.

Allocate Communications Buffer (123)

In Communication Manager/2, the maximum value of the requested buffer size is 64 KB minus 8 bytes (X'FFF8').

In Z and I Emulator for Windows, however, it is 64 KB minus 256 bytes (X'FF00').

ASCII Mnemonics

The following ASCII mnemonics are not supported in Z and I Emulator for Windows:

Mnemonics	Meaning
@A@N	Get Cursor
@A@O	Locate Cursor
@A@X	Hexadecimal
@A@Y	Cmd (Function) Key
@A@a	Destructive Backspace
@S@A	Erase EOL
@S@B	Field Advance
@S@C	Field Backspace
@S@D	Valid Character Backspace
@S@P	POR (For sending only)
@S@T	Jump to Task Manager
@/	Overrun of queue (Only in the Get Key function)

Get Request Completion (125)

Z and I Emulator for Windows does not support a blank or null session ID.

Appendix C. Notices

This information was developed for products and services offered in the United States. HCL may not offer the products, services, or features discussed in this information in other countries. Consult your local HCL representative for information on the products and services currently available in your area. Any reference to an HCL product, program, or service is not intended to state or imply that only that HCL product, program, or service may be used. Any functionally equivalent product, program or service that does not infringe any HCL intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-HCL product, program, or service.

HCL may have patents or pending patent applications covering subject matter described in this information. The furnishing of this information does not give you any license to these patents. You can send license inquiries, in writing, to:

HCL
330 Potrero Ave.
Sunnyvale, CA 94085
USA
Attention: Office of the General Counsel

HCL TECHNOLOGIES LTD. PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you..

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the information. HCL may make improvements and/or changes in the product(s) and/or program(s) described in this information at any time without notice.

Any references in this information to non-HCL documentation or non-HCL Web sites are provided for convenience only and do not in any manner serve as an endorsement of those documents or Web sites. The materials for those documents or Web sites are not part of the materials for this HCL product and use of those documents or Web sites is at your own risk.

HCL may use or distribute any of the information you provide in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

HCL
330 Potrero Ave.

Sunnyvale, CA 94085
USA
Attention: Office of the General Counsel

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by HCL under terms of the HCL Customer Agreement, HCL International Programming License Agreement, or any equivalent agreement between us.

The performance data discussed herein is presented as derived under specific operating conditions. Actual results may vary.

Information concerning non-HCL products was obtained from the suppliers of those products, their published announcements or other publicly available sources. HCL has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-HCL products. Questions on the capabilities of non-HCL products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to actual people or business enterprises is entirely coincidental.

Trademarks

HCL, the HCL logo, and hcl.com are trademarks or registered trademarks of HCL Technologies Ltd., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM® or other companies.