attachmate™

Worldwide Technical Support

HLLAPI Language Reference

Table of Contents

Purpose

This document is intended to assist customers wanting to enable existing or new automation software to work with a legacy application programming interface implemented in a current Attachmate emulator product: WinHLLAPI, EHLLAPI, Attachmate HLLAPI, Enterprise Access Library (EAL), PCSHLL (IBM PCOMM 4.01 EHLLAPI), or WD_API (Wall Data abstraction of HLLAPI).

Attachmate recommends that new automation programs be developed using EXTRA!'s COM (OLE Automation) interfaces. Only when a new automation program requires obscure capability not available in a COM solution should a legacy API be considered. In such situations, Attachmate recommends WinHLLAPI be given first preference, if only because it came about through an industry standardization effort. A second option would be EHLLAPI.

Introduction

An application programming interface, API, is typically provided in a software product to facilitate development of applications that automate tasks employing the software. For tasks that are highly repetitive, time-consuming or error-prone, automation can raise user job satisfaction, reduce operational costs, and improve service to customers.

The Attachmate High-Level Language Application Programming Interface (HLLAPI) is one such API. Introduced originally by Attachmate Corporation in the early 1990s in EXTRA! for Windows, Attachmate HLLAPI was a proprietary implementation of the already-popular programming interface, EHLLAPI, employed successfully throughout business and industry for a wide range of automation tasks.

The Attachmate HLLAPI interface provided through HLLAPI32.DLL is intended for use by applications written in C or C++. Due to the many language-specific conventions employed, it is quite awkward to attempt use of this API in other programming languages.

Header and lib files for EHLLAPI, WinHLLAPI, and Attachmate HLLAPI are distributed with EXTRA!.

HLLAPI Functions

Function cross-reference

This chapter describes each function provided in Attachmate's HLLAPI. The list below identifies the functions by number, name and entry-point name.

Number	Name	Entry point
1	Connect Presentation Space	HLL_ConnectPS
2	Disconnect Presentation Space	HLL_DisconnectPS
3	Send Key	HLL_SendKey
4	Wait	HLL_Wait
5	Copy Presentation Space	HLL_CopyPS
6	Search Presentation Space	HLL_SearchPS
7	Query Cursor Location	HLL_QueryCursorLocation
8	Copy Presentation Space to String	HLL_CopyPSToString
9	Set Session Parameters	HLL_SetHLLWinParameters
10	Query Sessions	HLL_QuerySessions
11	Reserve	HLL_Reserve
12	Release	HLL_Release
13	Copy OIA	HLL_CopyOIA
14	Query Field Attribute	HLL_QueryFieldAttribute
15	Copy String to Presentation Space	HLL_CopyStringToPS
16	Workstation Control	HLL_WSCtrl
18	Pause	HLL_Pause
20	Query System	HLL_QuerySystem
21	Reset System	HLL_ResetSystem
22	Query Session Status	HLL_QuerySessionStatus
23	Start Host Notification	HLL_StartHostNotification
24	Query Host Update	HLL_QueryHostUpdate
25	Stop Host Notification	HLL_StopHostNotification
30	Search Field	HLL_SearchField
31	Find Field Position	HLL_FindFieldPosition
32	Find Field Length	HLL_FindFieldLength
33	Copy String to Field	HLL_CopyStringToField
34	Copy Field to String	HLL_CopyFieldToString
40	Set Cursor	HLL_SetCursor
50	Start Keystroke Intercept	HLL_StartKeystrokeIntercept
51	Get Key	HLL_GetKey
52	Post Intercept Status	HLL_PostInterceptStatus
53	Stop Keystroke Intercept	HLL_StopKeystrokeIntercept
90	Send File	HLL_SendFile
91	Receive File	HLL_ReceiveFile
99	Convert Position or RowCol	HLL_Convert

What information is provided for each function?

For each HLLAPI function, the following information is presented:

- The function name and syntax used,
- Brief description of the function purpose,
- Prerequisites
- Applicable session parameters
- Call parameters
- Return parameters
- Notes or tips

Syntax
This section describes the correct syntax required to call the function.

Prerequisites
Many HLLAPI functions require another function to be called and successfully completed before the desired call is issued. If the prerequisites are not satisfied, an error code is returned. If None appears, no prerequisite calls are necessary.

Applicable session parameters
Function 9, "Set Session Parameters," allows an application program to set optional HLLAPI features, or session parameters. This section indicates whether any session parameters affect this function and, if so lists the applicable parameters and how they affect the function. If the function is not affected by any session parameters, None appears.

Call parameters
This area lists parameters that must be presented in a call statement when an application program can call a HLLAPI function.

Return parameters
Results returned to an application program by the functions are explained in this section.

Notes
This area presents guidelines and tips on how to use the function in an application program, along with technical information about the function.

Function 1: Connect Presentation Space

This function connects a HLLAPI application to a specified presentation space (PS). If the application already has a connection, the connected PS is automatically disconnected, and a new connection established. An application program must call this function before requesting any of the following-listed functions.

Number	Name
2	Disconnect Presentation Space
3	Send Key
4	Wait
5	Copy Presentation Space
6	Search Presentation Space
7	Query Cursor Location
8	Copy Presentation Space to String
11	Reserve
12	Release
13	Copy OIA
14	Query Field Attribute
15	Copy String to Presentation Space
30	Search Field
31	Find Field Position
32	Find Field Length
33	Copy String to Field
34	Copy Field to String
40	Set Cursor

You will be automatically disconnected from your currently connected PS when you connect to another PS using the same window handle (hWnd). Use separate window handles for each Connect Presentation Space call in order to connect to multiple sessions simultaneously.

Syntax

WORD HLL_ConnectPS(HWND hWnd, char cPSID)

Prerequisites

Sessions must be configured to be associated with a HLLAPI "short-name". End users must make this configuration on EXTRA!'s Global Preferences:Advanced dialog box.

Applicable session parameters

The following session parameters from Function 9 affect this function.

WRITE_SUPER (default)

This application requires write access and allows only supervisory applications to connect to its PS.

WRITE_WRITE

This application requires write access and allows other applications that have predictable behavior to connect to its PS.

WRITE_READ

This application requires write access and allows other applications to use read-only functions on its PS.

WRITE_NONE

This application requires exclusive access to its PS. No other applications may access its PS.

SUPER_WRITE

This supervisory application allows applications with write access to share the connected PS. The application program setting this parameter will not cause errors for other applications but will provide only supervisory-type functions.

WRITE_READ

This application requires read-only access and allows other applications that perform read-only functions to connect to its PS.

CONLOG (default)

When Function 1, "Connect Presentation Space," is called, the emulator session corresponding to the target PS does not become the active application. The calling application remains active. Likewise, when Function 2, "Disconnect Presentation Space," is called, the calling application remains active.

CONPHYS

Calling Function 1, "Connect Presentation Space," makes the emulator session corresponding to the target PS the active application (does a physical connect). Note that this parameter is honored only when there is host access software attached to the session. During Function 2, "Disconnect Presentation Space," the host access software becomes the active application.

Call parameters

An application program must pass the following parameters when calling this function:

Function	HLL_ConnectPS
hWnd	Window handle of the application
cPSID	1-character session short name which must be a letter of the alphabet

Return parameters

Result code

Function returns one of the following codes:

HLL_SUCCESS	The function succeeded; the PS is unlocked and ready for input.
HLL_INVALIDPSID	An invalid PSID (null or blank or unconfigured session) was specified.

HLL_PSLOCKED	The connection was successful, but the PS is busy or locked (all input is inhibited by XCLOCK or XSYSTEM).
HLL_SYSTEMERROR	A system error occurred; the function failed.

Example

```
/* Connect with session B */
WORD Result = HLL_ConnectPS(hWnd, 'B');
```

Notes

Unlike EHLLAPI or WinHLLAPI, an application can maintain connections to several HLLAPI sessions concurrently as long as the hWnd used for each conversation remains unique.

Function 2: Disconnect Presentation Space

This function disconnects an application from its currently connected PS and releases any PS keyboard reservation, but does not reset session parameters to defaults. After calling this function, the application cannot call functions that depend on connection to a PS.

An application automatically disconnects from the currently connected PS when it connects to another PS. After calling this function, you cannot call functions that depend on a connection to a PS. You will be automatically disconnected from your currently connected PS when you connect to another PS using the same window handle (hWnd). Use separate window handles for each Connect Presentation Space call in order to connect to multiple sessions simultaneously.

Syntax

```
WORD HLL_DisconnectPS (HWND hWnd);
```

Prerequisites

Function 1, "Connect Presentation Space."

Applicable session parameters

The following session parameter from Function 9 affects this function.

CONPHYS

If set (as opposed to default CONLOG), the calling application becomes activated when HLLAPI function 2 is called.

Call parameters

An application program must pass the following parameters when calling this function:

Function	HLL_DisconnectPS
hWnd	Window handle of the application

Return parameters

Result code

Function returns one of the following codes:

HLL_SUCCESS	The function succeeded.
HLL_INVALIDPSID	Not connected.
HLL_SYSTEMERROR	A system error occurred; the function failed.

Example

```
WORD Result = HLL_DisconnectPS(hWnd);
```

Notes

This function only logically disconnects an application from an EXTRA! session. It does not signal the end of EHLLAPI interaction by the application. In contrast, a call to function 21, “Reset System,” frees resources used by EXTRA! and allows disconnected session(s) to close when the application exits.

Function 3: Send Key

This function sends a string of up to 255 keystrokes to the currently connected PS. The session cannot receive keystrokes unless the keyboard is unlocked. After the first AID key is processed by the function, keystrokes are no longer accepted and the rest of the string is ignored.

It is possible to represent all necessary keystrokes, including special function keys in ASCII, by using an escape character (the default value is @) followed by the appropriate key code. Appendix B, "Keyboard Mnemonics," provides a complete list of these key codes.

HLLAPI changes the cursor position to the position immediately following the entered string.

Syntax

WORD HLL_SendKey (HWND hWnd, LPSTR lpszKeys);

Prerequisites

Function 1, "Connect Presentation Space."

The keyboard must be unlocked before keystrokes will be accepted.

Applicable session parameters

The following session parameters from Function 9 affect this function.

ESC= char

Specifies the escape character for keystroke mnemonics ("@" is the default). Blank is not a valid escape value.

AUTORESET (default)

Attempts to reset inhibited conditions by adding the RESET prefix to all keystroke strings sent.

NORESET

Does not add RESET prefix to key strings.

Call parameters

An application program must pass the following parameters when calling this function:

Function	HLL_SendKey
hWnd	Window handle of the application
lpszKeys	A string of maximum 255 characters (keystrokes) to be sent to the host PS, terminated with a null character.

Return parameters

Result code

Function returns one of the following codes:

HLL_SUCCESS	The function succeeded.
HLL_INVALIDPSID	Not connected.
HLL_INVALIDPARAMETER	The lpszKeys string was empty or the given length was less than zero or greater than 255.
HLL_SESSIONOCCUPIED	The host system session was busy; not all of the keystrokes could be sent.
HLL_PSLOCKED	Input to the target session was inhibited or rejected; all of the keystrokes could not be sent.
HLL_SYSTEMERROR	A system error occurred; the function failed.

Example

```
/* Send "Hello" followed by Enter keystroke */
WORD Result = HLL_SendKey(hWnd, "Hello@E");
```

Notes

- For increased performance, an application may send entire strings using Function 33, "Copy String to Field," or Function 15, "Copy String to Presentation Space," rather than using this function; however, only function 3 may send escape sequences such as ENTER or PF3.
- If the keystroke string is longer than 255 characters (which is the Send Key function's limit), use multiple calls to the Send Key function.

Function 4: Wait

This function provides current status of XCLOCK or XSYSTEM conditions of the OIA. (Function 9, "Set Session Parameters," allows a program to vary the amount of time this function will wait for the OIA to clear.)

Because host applications are so different and a terminal cannot determine when a host application is ready for input, the HLLAPI application must determine when the host is ready for more input. The Wait function is not a good method for determining when the host is ready for input. This function is provided to determine if the terminal session can accept keystrokes (using "Send Key" or a copy function). To determine when the host session is ready, the application should search the screen for key fields, usually near the bottom of the screen. Another method is to query the cursor position until it is located at the correct field. Or an application could use this function repetitively until is returns successfully for a certain period of time, perhaps one or two seconds.

If the application program is already in a Wait, Pause, Get Key, or synchronous file transfer, the request for another delay is rejected.

Syntax

```
WORD HLL_Wait (HWND hWnd);
```

Prerequisites

Function 1, "Connect Presentation Space."

Applicable session parameters

The following session parameters from Function 9 affect this function.

TWAIT (default)

The function waits up to one minute before it times out waiting for XCLOCK or XSYSTEM (3270) or II (5250) to clear.

LWAIT

The function waits for XCLOCK or XSYSTEM (3270) or II (5250) to clear before returning to the calling application.

NWAIT

The function does not wait but returns immediately with busy / inhibited status.

Call parameters

An application program must pass the following parameters when calling this function:

Function	HLL_Wait
hWnd	Window handle of the application

Return parameters

Result code

Function returns one of the following codes:

HLL_SUCCESS	The keyboard is unlocked and ready for input.
HLL_INVALIDPSID	Not connected.
HLL_TIMEOUT	The function timed out while still in XCLOCK or XSYSTEM.
HLL_PSLOCKED	The keyboard is locked.
HLL_SYSTEMERROR	A system error occurred; the function failed.
HLL_RESOURCEUNAVAILABLE	The client application window is already waiting, pausing, getting a key, operating a synchronous file transfer, and so forth.

Example

```
WORD Result = HLL_Wait(hWnd);
```

Notes

- This function can be used together with a function like Function 6, "Search Presentation Space," to determine when the host is ready for the next input.
- The HLLAPI application should consider relative machine speed. For example, a host may complete its task during a Wait on a slow machine, but a faster machine may need another approach, as noted earlier.

Function 5: Copy Presentation Space

This function copies the currently connected PS to a string allocated in the calling application.

Syntax

WORD HLL_CopyPS (HWND hWnd, LPSTR lpBuffer);

Prerequisites

Function 1, "Connect Presentation Space."

Applicable session parameters

The following session parameters from Function 9 affect this function.

XLATE = 1 (default)

Translates strings to be copied to the PS, as well as search strings, into 3270 display codes (EBCDIC). Translates strings copied from the PS into ASCII.

NOXLATE = 2

Skips translation. Allows an application to copy 3270 display codes directly to and from the PS, and to search directly for 3270 display-code strings.

NOATTRB (default)

Attribute bytes and other characters not displayable in ASCII are translated into blanks.

ATTRB

Attribute bytes and other characters not displayable in ASCII are not translated.

EAB

Extended Attribute Bytes (EABs) are copied. Two characters are placed in the application data string for each one that appears in the PS. The EAB is the second character. To accommodate this, the application program must allocate a data string that is twice the number of displayable characters to be copied from the presentation space of the current display model.

NOEAB (default)

EABs are not copied.

Call parameters

An application program must pass the following parameters when calling this function:

Function	HLL_CopyPS
hWnd	Window handle of the application

lpBuffer A string large enough to accommodate data from the current PS display

Model (including EABs if requested). See chart below.

Model number	Data string length required
2	1920 (3840 with EABs)
3	2560 (5120 with EABs)
4	3440 (6880 with EABs)
5	3564 (7128 with EABs)

Return parameters

Data string

Function replaces content of call parameter lpBuffer with text and, if requested, extended attribute bytes from the presentation space.

Refer to Appendix D, "Extended Attributes," for information on EAB interpretation.

Result code

Function returns one of the following codes:

HLL_SUCCESS	The function succeeded.
HLL_INVALIDPSID	No connected.
HLL_SESSIONOCCUPIED	The PS contents were copied; the PS was waiting for host system response (XCLOCK or XSYSTEM).
HLL_PSLOCKED	The PS was copied; the keyboard was locked.
HLL_SYSTEMERROR	A system error occurred; the function failed.

Example

```
/* Reserve string for text from Model 2 screen w/o EABs (1920
bytes */
char HllDataStr[2000];
WORD Result = HLL_CopyPS(hWnd, HllDataStr);
```

Notes

- Use this function only when the entire PS is needed; otherwise, use Function 8, "Copy Presentation Space to String," or Function 34, "Copy Field to String."
- Use Function 10, "Query Sessions," or Function 22, "Query Session Status," to check host session PS size.

Function 6: Search Presentation Space

This function searches the currently connected PS for first or last occurrence of specified text.

This function is useful for determining whether a specific host panel is present. For example, if the application is expecting a prompt before sending data, this function will search for the message or string before moving on. If the prompt or message is not found, the application program can call Function 18, "Pause," or Function 24, "Query Host Update," and continue to call Function 6 until the string is found.

Syntax

```
DWORD HLL_SearchPS(HWND hWnd, LPSTR lpsSearchString,
                                    WORD wStringLength, WORD wPSP);
```

Prerequisites

Function 1, "Connect Presentation Space."

Applicable session parameters

The following session parameters from Function 9 affect this function.

SRCHALL and SRCHFRWD (default)

The function scans the entire PS for the first occurrence of the specified string.

SRCHALL and SRCHBKWD

The function scans the entire PS for the last occurrence of the specified string.

SRCHFROM and SRCHFRWD

The function scans the PS from the specified PS position for the first occurrence of the string.

SRCHFROM and SRCHBKWD

The function scans the PS from the specified PS position for the last occurrence of the string.

XLATE = 1 (default)

Translates strings to be copied to the PS, as well as search strings, into 3270 display codes (EBCDIC). Translates strings copied from the PS into ASCII.

NOXLATE = 2

Skips translation. Allows an application to copy 3270 display codes directly to and from the PS, and to search directly for 3270 display-code strings.

Call parameters

An application program must pass the following parameters when calling this function:

Function	HLL_SearchPS
hWnd	Window handle of the application
lpsSearchString	Text to be searched for in the PS
wStringLength	This specifies the length of the search string. It is used if HLLWIN_NOXLATE is set; otherwise, it is ignored.
wPSP	Start position where the search function is to begin (SRCHFRWD) or to end (SRCHBKWD). This parameter is ignored if SRCHALL is set.

Return parameters

Function returns a double word (4 bytes).

PS Position

The high-order word of the return value contains the PS position where specified text was found, or 0 if the text was not found.

Result code

The low-order word of the return value contains one of the following codes:

HLL_SUCCESS	The function succeeded; the string was found.
HLL_INVALIDPSID	Not connected.
HLL_INVALIDPARAMETER	An error was made in specifying parameters (for example, LLWIN_NOXLATE is set but wStringLength=0).
HLL_INVALIDPSPOSITION	An invalid PS position was specified.
HLL_SYSTEMERROR	A system error occurred; the function failed.
HLL_SEARCHSTRINGNOTFOUND	The search string was not found.

Example

```
/* Search for: "Hello" starting at PS position 199 */
DWORD Result = HLL_SearchPS(hWnd, "Hello", 5, 199);
```

Notes

- The SRCHFROM option is useful when you are searching for a string that may occur several times.
- The search carried out by this function is case-sensitive.
- To determine when the host is ready for input, the application should search the screen for key fields, usually near the bottom of the screen.

Function 7: Query Cursor Location

This function returns the position of the cursor in the currently connected PS.

Syntax

DWORD HLL_QueryCursor (HWND hWnd);

Prerequisites

Function 1, "Connect Presentation Space."

Applicable session parameters

None.

Call parameters

An application program must pass the following parameters when calling this function:

Function	HLL_QueryCursorLocation
hWnd	Window handle of the application

Return parameters

Function returns a double word (4 bytes).

PS Position

The high-order word of the return value contains the PS position of the cursor.

Result code

The low-order word of the return value contains one of the following codes:

HLL_SUCCESS	The function succeeded.
HLL_INVALIDPSID	Not connected.
HLL_SYSTEMERROR	A system error occurred; the function failed.

Example

```
DWORD Result = HLL_QueryCursorLocation(hWnd);
```

Notes

- This function is one method of determining whether a host session is at a particular screen.

- To make this determination, the application can repeatedly query cursor position until it is located at the correct field.

Function 8: Copy Presentation Space to String

This function copies all or part of the currently connected PS to a string allocated in the calling application.

Syntax

```
WORD HLL_CopyPSToString (HWND hWnd, LPSTR lpBuffer,
                         WORD wBufferLength, WORD wPSP);
```

Prerequisites

Function 1, "Connect Presentation Space."

Applicable session parameters

The following session parameters from Function 9 affect this function.

XLATE = 1 (default)

Translates strings to be copied to the PS, as well as search strings, into 3270 display codes (EBCDIC). Translates strings copied from the PS into ASCII.

NOXLATE = 2

Skips translation. Allows an application to copy 3270 display codes directly to and from the PS, and to search directly for 3270 display-code strings.

NOATTRB (default)

Attribute bytes and other characters not displayable in ASCII are translated into blanks.

ATTRB Attribute bytes and other characters not displayable in ASCII are not translated.

EAB

Extended Attribute Bytes are copied. Two characters are placed in the application data string for each one that appears in the PS. The EAB is the second character. To accommodate this, the application program must allocate a data string that is twice the number of displayable characters to be copied. For example, 160 bytes should be allotted to copy the first 80 characters with EABs.

NOEAB (default)

Extended Attribute Bytes are not copied.

Call parameters

An application program must pass the following parameters when calling this function:

Function	HLL_CopyPSToString
hWnd	Window handle of the application
lpBuffer	A string of sufficient size to hold data requested from the PS, including EABs if requested
wBufferlength	The number of characters allocated in Data string.
WPSP	The PS position where the copying should begin.

Return parameters

Data string

Function replaces content of call parameter lpBuffer with text from the presentation space.

Refer to Appendix D, "Extended Attributes," for information on EAB interpretation.

Result code

Function returns one of the following codes:

HLL_SUCCESS	The PS contents were copied to the string; the PS was active and the keyboard was unlocked.
HLL_INVALIDPSID	Not connected.
HLL_INVALIDPARAMETER	An error was made in specifying the string length.
HLL_SESSIONOCCUPIED	The PS contents were copied; the currently connected PS was waiting for the host response.
HLL_PSLOCKED	The PS was copied; the keyboard was locked.
HLL_INVALIDPSPOSITION	An invalid PS position was specified.
HLL_SYSTEMERROR	A system error occurred; the function failed.

Example

```
/* Start position to copy */
WORD PsPos = 199;
/* Buffer for returned data */
char DataStr[5];
/* Length of string to copy */
WORD DataLn = 5;
WORD Result = HLL_CopyPSToString(hWnd, DataStr, DataLn, PsPos);
```

Function 9: Set Session Parameters

This function sets session parameters in HLLAPI for the connected presentation space. Parameters set with this function affect many other HLLAPI functions, as noted in individual function descriptions ("Applicable session parameters") and in descriptions of this function's call parameters.

Session parameters set with function 9 remain in effect until one of the following occurs:

- Function 21, "Reset System," which resets the session parameters to default values
- A new value is specified by a second function 9 call
- The HLLAPI client application program terminates

Syntax

WORD HLL_SetHLLWinParameters (HWND hWnd, LPHLLPARAMS lpHLLParams);

Prerequisites

None.

Applicable session parameters

None.

Call parameters

An application program must pass the following parameters when calling this function:

Function	HLL_SetHLLWinParameters
hWnd	Window handle of the application
lpHLLParams	A structure containing desired values of session parameters. (Format below)

Parameter structure

The session-parameter structure is comprised of 13 bytes organized as shown:

Byte	Description
0	Attribute control (`ATTRB or NOATTRB`)
1	Autoreset control (`AUTORESET or NOAUTORESET`)
2	Connect type control (`CONLOG or CONPHYS`)
3	Extended attribute control (`EAB or NOEAB`)
4	Escape character (`default '@'`)

5	Pause control (IPAUSE or FPAUSE)
6	Search Origin control (SRCHALL or SRCHFROM)
7	Search Direction control (SRCHFRWD or SRCHBKWD)
8-9	Timeout control (0 .. 64k, a binary word)
10	Reserved
11	Wait control (TWAIT, LWAIT, NWAIT)
12	Xlate control (XLATE or NOXLATE)

For formal declarations of these parameters, see header file HLLAPI.H.

NOTE: It is recommended an application call function HLL_QueryHLLWinParameters to initialize the structure with current values before making changes.

Copy parameters

The following session parameters affect all copy functions.

ATTRB = 1

EBCDIC characters that cannot be translated to displayable ASCII characters are not translated.

NOATTRB = 2 (default)

EBCDIC characters that cannot be translated to displayable ASCII characters are translated to blanks (0x20).

EAB = 1

Extended Attribute Bytes are copied along with data.

NOEAB = 2 (default)

EABs are not copied (data only).

XLATE = 1 (default)

Translates strings to be copied to the PS, as well as search strings, into 3270 display codes (EBCDIC). Translates strings copied from the PS into ASCII.

NOXLATE = 2

Skips translation. Allows an application to copy 3270 display codes directly to and from the PS, and to search directly for 3270 display-code strings.

Connect parameters

The following session parameters affect Function 1, "Connect Presentation Space," and Function 2, "Disconnect Presentation Space."

CONLOG = 1 (default)

When Function 1, "Connect Presentation Space," is called, the emulator session corresponding to the target PS does not become the active application. The calling application remains active. Likewise, when Function 2, "Disconnect Presentation Space," is called, the calling application remains active.

CONPHYS = 2

Calling Function 1, "Connect Presentation Space," makes the emulator session corresponding to the target PS the active application (does a physical connect). Note that this parameter is honored only when there is host access software attached to

the session. During Function 2, "Disconnect Presentation Space," the host access software becomes the active application.

Esc/Reset parameters

The following session parameters affect Function 3, "Send Key," and Function 51,"Get Key."

ESC= char

Specifies the escape character for keystroke mnemonics ("@" is the default). Blank is not a valid escape value.

AUTORESET = 1 (default)

Attempts to reset all inhibited conditions by adding the prefix RESET to all keystroke strings sent using Function 3, "Send Key."

NORESET = 2

Does not add RESET prefix to function 3 key strings.

Search parameters

The following session parameters affect all search functions.

SRCHALL = 1 (default)

Scans the entire PS or field.

SRCHFROM = 2

Starts the scan from a specified location in the PS or field.

SCRCHFRWD = 1 (default)

Performs the scan in an ascending direction.

SRCHBKWD = 2

Performs the scan in a descending direction through the PS or field.

Wait parameters

The following session parameters affect Function 4, "Wait," and Function 51, "Get Key."

TWAIT = 1 (default)

For Function 4, "Wait," TWAIT waits up to a minute before timing out on XCLOCK or XSYSTEM.

For Function 51, "Get Key," TWAIT does not return control to the HLLAPI client application program until it has intercepted a key (a normal or AID key, based on the option code specified under Function 50, "Start Keystroke Intercept").

LWAIT = 2

For Function 4, "Wait," LWAIT waits until XCLOCK / XSYSTEM clears. This option is not recommended because XSYSTEM or permanent XCLOCK will prevent control being returned to the application.

For Function 51, "Get Key," LWAIT does not return control to your application until it has intercepted a key. The intercepted key could be a normal or AID key, based on the option specified under Function 50, "Start Keystroke Intercept."

NWAIT = 3

For Function 4, "Wait," NWAIT checks status and returns immediately (no wait).

For Function 51, "Get Key," NWAIT returns code 25 (keystroke not available) if nothing matching the option specified under Function 50, "Start Keystroke Intercept," is queued.

Pause parameters

The following session parameters affect Function 18, "Pause," determining the type of pause to perform.

NOTE: An application can make multiple Function 23 calls, and an event satisfying any of the calls will interrupt the pause.

FPAUSE = 1 (default)

Full-duration pause. Control returns to the calling application when the number of half-second intervals specified in the Function 18 call have elapsed.

IPAUSE = 2

Interruptible pause; Control returns to the calling application when a system even specified in a preceding Function 23, "Start Host Notification," call has occurred, or the number of half-second intervals specified in the Function 18 call have elapsed.

Timeout parameter

The following session parameter affects Function 90, "Send File," and Function 91, "Receive File."

TIMEOUT= n

Specifies the number of 10 millisecond intervals a file-transfer operation should be allowed to run before CTRL BREAK is issued to terminate it. Zero means CTRL BREAK will not be issued.

Return parameters

Result code

Function returns one of the following codes:

HLL_SUCCESS	The function succeeded.
HLL_INVALIDPARAMETER	One or more parameter values were unrecognized; all recognized values were accepted.
HLL_SYSTEMERROR	A system error occurred; the function failed.

Example

```
/* Allocate session-parameter structure */
HLLPARAMS Params; /* Refer to HLLAPI.H for details */
/* Update session parameters */
WORD ResultO = HLL_SetHLLWinParameters(hWnd, &Params);
```

Function 10: Query Sessions

This function returns summary information about each HLLAPI-configured session. The information is returned in a 12-byte structure for each session.

Syntax

```
DWORD HLL_QuerySessions(HWND hWnd, LPSESSIONS lpSessions,
                        WORD wSessionState, WORD wNumberOfSessions);
```

Prerequisites

None.

Applicable session parameters

None.

Call parameters

An application program must pass the following parameters when calling this function:

Function	HLL_QuerySessions
hWnd	Window handle of the application
lpSessions	An array of one or more 12-byte session structures.
wSessionState	An integer specifying the state of session(s) for which information is wanted HLL_QUERYSESSIONSCONFIGURED HLL_QUERYSESSIONSOPENED HLL_QUERYSESSIONSPOWERED
wNumberOfSessions	The number of session structures in call parameter *lpSessions*.

Return parameters

Function returns a double word (4 bytes).

Sessions started

The high-order word of the return value contains the count of the number of sessions found that met the specified session-state criterion.

Result code

The low-order word of the return value contains one of the following codes:

HLL_SUCCESS	The function succeeded.
HLL_INVALIDPARAMETER	null–lpSessions or zero wNumberOfSessions.
HLL_SYSTEMERROR	A system error occurred; the function failed.

Session information

Function replaces content of call parameter Sessions with information about sessions meeting the specified session-state criterion, filling each structure as follows:

Byte	Description
1	Session short name.
2–9	Session long name (first 8 characters of the "edp" filename)
10	A fixed constant byte value (‘H’)
11–12	PS size as a binary word.

Example

```
/* Allocate space for as many as 26 sessions */
SESSIONS Sessions[26];
/* Get information about opened sessions */
DWORD Result = HLL_QuerySessions(hWnd, Sessions,
HLL_SESSIONOPENED, 26);
```

Function 11: Reserve

This function locks the currently connected PS. Locking the PS prevents another application program or terminal operator from entering data into it. Once the PS is locked, it is not accessible until it is unlocked.

The PS can be unlocked with Function 12, "Release"; Function 21, "Reset System"; Function 2, "Disconnect Presentation Space"; or Function 1, "Connect Presentation Space." Function 1 performs an implicit disconnect. (Terminating a session with Task Manager also unlocks it.)

This function is useful for preventing users from gaining access to the session while an application program sends a series of transactions to the host.

Syntax

WORD HLL_Reserve (HWND hWnd);

Prerequisites

Function 1, "Connect Presentation Space."

Applicable session parameters

None.

Call parameters

An application program must pass the following parameters when calling this function:

Function	HLL_Reserve
hWnd	Window handle of the application

Return parameters

Result code

Function returns one of the following codes:

HLL_SUCCESS	The function succeeded.
HLL_INVALIDPSID	Not connected.
HLL_SYSTEMERROR	A system error occurred; the function failed.

Example

```
WORD Resullt = HLL_Reserve(hWnd);
```

Function 12: Release

This function unlocks a PS that was reserved using Function 11, "Reserve." The target is the currently connected PS.

Release also occurs automatically when the client application program calls Function 2, "Disconnect Presentation Space"; Function 1, "Connect Presentation Space"; Function 21, "Reset System"; or terminates, or the session itself is terminated.

Because release occurs automatically on disconnect, it is not crucial that you use the Release function whenever you end an application.

Syntax

```
WORD HLL_Release (HWND hWnd);
```

Prerequisites

Function 1, "Connect Presentation Space."

Applicable session parameters

None.

Call parameters

An application program must pass the following parameters when calling this function:

Function	HLL_Release
hWnd	Window handle of the application

Return parameters

Result code

Function returns one of the following codes:

HLL_SUCCESS	The function succeeded.
HLL_INVALIDPSID	Not connected.
HLL_SYSTEMERROR	A system error occurred; the function failed.

Example

```
WORD Resullt = HLL_Release(hWnd);
```

Function 13: Copy OIA

This function returns the contents of the OIA from the currently connected PS. The length of the OIA data does not change with the terminal model.

Syntax

WORD HLL_CopyOIA (HWND hWnd, LPSTR lpOIA);

Prerequisites

Function 1, "Connect Presentation Space."

Applicable session parameters

None.

Call parameters

The following session parameters from Function 9 affect this function.

Function	HLL_CopyOIA
hWnd	Window handle of the application
lpOIA	A pre-allocated 103-byte data string

Return parameters

OIA data

Function replaces content of call parameter lpOIA with data from the OIA for the currently-connected PS, organized as follows:

Byte	Description
1	The OIA Format Byte for the host access program.
2–81	These bytes contain the untranslatable image of the OIA in hexadecimal codes.
82–103	The OIA bit group.

Detailed explanation of information contained in this string is given in Appendix C, "Interpreting the Returning Data String for Function 13."

Result code

Function returns one of the following codes:

HLL_SUCCESS	The function succeeded.
HLL_INVALIDPSID	Not connected.
HLL_INVALIDPARAMETER	A parameter error occurred; no OIA data was returned.
HLL_SESSIONOCCUPIED	The OIA data was returned; the target PS is busy.

HLL_PSLOCKED	The OIA data was returned; the target PS is locked.
HLL_SYSTEMERROR	A system error occurred; no OIA data was returned.

Example

```
char HllDataStr[103];
/* Length of allocated data area */
WORD Result = HLL_CopyOIA(hWnd, HllDataStr);
```

Function 14: Query Field Attribute

This function returns the attribute byte for the field at the specified PS position in the currently-connected presentation space. If extended attribute bytes (EABs) are enabled and present, the function returns the EAB value as well.

Syntax

DWORD HLL_QueryFieldAttribute (HWND hWnd, WORD wPSP);

Prerequisites

Function 1, "Connect Presentation Space."

Applicable session parameters

None.

Call parameters

An application program must pass the following parameters when calling this function:

Function	HLL_QueryFieldAttribute
hWnd	Window handle of the application
wPSP	The PS position for which field information is wanted

.

Return parameters

Function returns a double word (4 bytes).

Result code

The low-order word of the return value contains one of the following codes:

HLL_SUCCESS	The function succeeded.
HLL_INVALIDPSID	Not connected.
HLL_INVALIDPSPOSITION	An invalid PS position was specified.
HLL_SYSTEMERROR	A system error occurred; the function failed.
HLL_UNFORMATTEDHOSTPS	The host PS was unformatted.

Attribute value

The low-order byte of the high-order word contains the value of the attribute byte that begins the field pointed to by call parameter wPSP. If zero, the PS is unformatted and no attribute can be returned.

3270 Field attribute

Bit	Meaning
0-1	Both = 1, field attribute value
2	0 = unprotected; 1 = protected
3	0 = alphanumeric; 1 = numeric only
4-5	00 = normal intensity, not pen detectable 01 = normal intensity, pen detectable 10 = high intensity, pen selectable 11 = nondisplay, not pen detectable
6	Reserved
7	0 = field has not been modified; 1 = field has been modified

5250 Field attribute

Bit	Meaning
0	0 = nonfield attribute; 1 = field attribute
1	0 = nondisplay; 1 = display
2	0 = unprotected; 1 = protected
3	0 = normal intensity; 1 = high intensity
4-6	000 = alphameric data; all characters available 001 = alphabetic only, u/c and l/c, comma, period, hyphen, blank and Dup available 010 = numeric shift; automatic shift for number 011 = numeric only: 0-9, comma, period, plus, minus, blank and Dup available 101 = numeric only: 0-9 or Dup available 110 = magnetic strip reading device data only 111 = signed numeric data: 0-9, plus, minus and Dup are available
7	0 = field has not been modified; 1 = field has been modified

Extended attribute value

The high-order byte of the high-order word contains the Extended Attribute Byte for the specified PS position, if EABs are enabled and present. (Refer to Appendix D, "Extended Attributes," for information on interpreting this value.

Example

```
/* Query field attribute at position 199 */
DWORD Result = HLL_QueryFieldAttribute(hWnd, 199);
```

Function 15: Copy String to Presentation Space

This function copies a string directly into the currently connected PS at the specified location. When the copy operation is complete, the cursor's physical location remains unchanged.

The data string to be copied cannot be any larger than the size of the designated writable area or field. Unprintable characters in the string are translated into blanks in the host system session.

Syntax

```
WORD HLL_CopyStringToPS(HWND hWnd, LPSTR lpString,
                        WORD wStringLength, WORD wPSP);
```

Prerequisites

Function 1, "Connect Presentation Space."

Applicable session parameters

The following session parameters from Function 9 affect this function.

XLATE = 1 (default)

Translates strings to be copied to the PS, as well as search strings, into 3270 display codes (EBCDIC). Translates strings copied from the PS into ASCII.

NOXLATE = 2

Skips translation. Allows an application to copy 3270 display codes directly to and from the PS, and to search directly for 3270 display-code strings.

EAB

Extended Attribute Bytes are copied. Two characters are copied from the application data string for each position in the PS. The EAB is the second character.

NOEAB (default)

Extended Attribute Bytes are not present.

Call parameters

An application program must pass the following parameters when calling this function:

Function	HLL_CopyStringToPS
hWnd	Window handle of the application

lpString	ASCII text and, if requested, EABs to be copied into the PS. Refer to Appendix D, “Extended Attributes,” for information on EAB format.
wStringLength	Data string length.
WPSP	Position of the PS where function is to begin copying data.

Note: This function cannot send keyboard mnemonics for host commands.

Return parameters

Result code

Function returns one of the following codes:

HLL_SUCCESS	The function succeeded.
HLL_INVALIDPSID	Not connected.
HLL_INVALIDPARAMETER	An invalid parameter string was specified (the string contains no characters or the given string length parameter was zero).
HLL_PSLOCKED	The target PS is protected or inhibited, or illegal data was sent to the target PS (for example, field attribute byte when HLLWIN_XLATE is set).
HLL_DATATRUNCATED	The copy was completed but the size of the string is longer than the available PS space. The part of the string that extends beyond the PS will be truncated.
HLL_INVALIDPSPOSITION	An invalid PS position was specified.
HLL_SYSTEMERROR	A system error occurred; the function failed.

Example

```
/* Copy "Hello World" starting at position 199 of PS */
WORD Result = HLL_CopyStringToPS(hWnd, "Hello World", 11, 199);
```

Notes

To copy data to the current PS position, use Function 7, “Query Cursor Location,” to obtain the PS position, then use that value as the PS position calling parameter of this function.

Function 16: WSCtrl

This function allows an application to control product features such as configuration, layouts and emulator window state.

Syntax

```
WORD HLL_WSCtrl (HWND hWnd, WORD wOption,
                 LPVOID lpvState, WORD wStateLength);
```

Prerequisites

None.

Applicable session parameters

None.

Call parameters

An application program must pass the following parameters when calling this function:

Function	HLL_WSCtrl
hWnd	Window handle of the application
wOption	number specifying which WSCtrl subfunction is requested: HLL_WSCTRLOPENLAYOUT HLL_WSCTRLQUERYLAYOUT HLL_WSCTRLQUERYEXTRADIRECTORY HLL_WSCTRLEMULATORHANDLE HLL_WSCTRLSTARTEMULATOR HLL_WSCTRLSTOPEMULATOR HLL_WSCTRLTERMINALON HLL_WSCTRLTERMINALOFF HLL_WSCTRLBLOCKEMULATORUPDATES HLL_WSCTRLALLOWEMULATORUPDATES
lpvState	Additional controls pertaining to the requested option. (See following)
wStateLength	2-byte integer specifying the number of bytes in *lpvState* of significance

Control structures

Format and organization of lpvState depends on the Option requested.

Option	Data-string format
Open layout	Null-terminated ASCII text giving location and name of the layout file.
Query layout	A pre-allocated string to receive location and name of current layout.
Emulator handle	Address of a WORD, the low byte of which is the byte value of the

	session short name on entry; the WORD on return is the emulator window handle
Start emulator	12-byte EMULATORCONTROL structure providing the short name and window state of the session to be opened. (See format below)
Stop emulator	NA (Data string and data length are ignored for this option)
Terminal On	NA (Data string and data length are ignored for this option)
Terminal Off	NA (Data string and data length are ignored for this option)
Allow updates	NA (Data string and data length are ignored for this option)
Block updates	NA (Data string and data length are ignored for this option)
Query Extra Directory	A buffer which will contain the EXTRA! install directory on return.

Start Emulator control structure

When the requested option is Start Emulator, the following control structure is required.

Byte	Description
1	Session short name.
2	Visibility (‘N’ = normal; ‘I’ = icon; ‘M’ = maximized; ‘H’ = hidden)
3	Reserved
4	Case (‘U’ = uppercase; ‘M’ = upper/lower)
5-6	Left coordinate of normal session window
7-8	Bottom coordinate of normal session window
9-10	Right coordinate of normal session window
11-12	Top coordinate of normal session window

Return parameters

Result code

Function returns one of the following codes:

HLL_SUCCESS	The function succeeded.
HLL_INVALIDPARAMETER	An invalid parameter string (for example, lpString null or wStringlength=0) was specified.
HLL_SYSTEMERROR	A system error occurred; the function failed.
HLL_INVALIDPSID	Not connected.
HLL_RESOURCEUNAVAILABLE	The requested PS is being used by another client application window connection.
HLL_NOEMULATORATTACHED	This call requires an emulator attached to the session.
HLL_WSCTRLFAILURE	The workstation control function failed.

Example

```
/* Point control structure to desired layout file */
char Layout[] = "C:\\Program Files\\E!PC\\Sessions\\Monitor.ELF";
/* Open layout */
WORD Result = HLL_WSCtrl(hWnd, HLL_WSCTRLOPENLAYOUT, Layout,
                  strlen(Layout));

/* Get window handle to session B */
WORD wHandle = MAKEWORD('B', 0);
WORD Result = HLL_WSCtrl(hWnd, HLL_WSCTRLEMULATORHANDLE, wHandle, 4);
```

Function 18: Pause

This function waits a specified amount of time or until a host-initiated update occurs.

If the client application program is already in a Wait, Pause, Get Key, or synchronous file transfer delay, the request for another delay is rejected.

Syntax

```
WORD HLL_Pause (HWND hWnd, WORD wDuration);
```

Prerequisites

Function 23, "Start Host Notification," must be called if the application program uses session parameter IPAUSE."

Applicable session parameters

The following session parameters from Function 9 affect this function.

FPAUSE (default)

The function waits the amount of time specified if session parameter FPAUSE is in effect.

IPAUSE

The function waits until a specified host update occurs if session parameter IPAUSE is set and the application has called Function 23, "Start Host Notification. The application must call Function 24, "Query Host Update," before setting the next pause; otherwise, the next pause will be immediately satisfied by the pending event.

Call parameters

An application program must pass the following parameters when calling this function:

Function	HLL_Pause
hWnd	Window handle of the application
wDuration	The pause duration in 1/2-second multiples.

Return parameters

Result code

Function returns one of the following codes:

HLL_SUCCESS	The pause duration has expired.

HLL_SYSTEMERROR	A system error occurred; the function failed.
HLL_RESOURCEUNAVAILABLE	Another pause, wait, get key, or synchronous file transfer is in progress for the specified client application window.
HLL_HOSTSESSIONUPDATE	A host session monitored by the pausing client application has been updated; use Function 24, “Query HostUpdate,” for more information.

Example

```
/* Wait for 10 sec. or until interrupted */
WORD Result = HLL_Pause(hWnd, 20);
```

Notes

Instead of using interruptible pauses, the client application can establish the host monitor (Function 23, “Start Host Notification”) specifying the windows message service. HLLAPI will then post XM_SESSIONUPDATE messages to the hWnd when host session updates occur. See Appendix E, “Attachmate HLLAPI Messages,” and Function 23 for details.

This function returns the cursor to its class shape. For example if you define the cursor as an hourglass and perform a lengthy operation, the first time you call this function, it will return your cursor shape to an arrow.

Function 20: Query System

This function returns information about system state that may be useful for determining the cause of a result code 9 being received from some other function call.

Syntax

WORD HLL_QuerySystem (HWND hWnd, LPSYSTEM lpSystem);

Prerequisites

None.

Applicable session parameters

None.

Call parameters

An application program must pass the following parameters when calling this function:

Function	HLL_QuerySystem
hWnd	Window handle of the application
lpSystem	A 9-byte structure to receive system information.

Return parameters

System information

Function replaces content of call parameter lpSystem with information about the system state, organized as follows:

Byte	Description
1	Month
2	Day
3–4	Year
5	Short name of currently-connected PS
6-9	Extended error code 1 (system component, printable ASCII)

Result code

Function returns one of the following codes:

HLL_SUCCESS	The function succeeded.
HLL_SYSTEMERROR	A system error code for the specified client application.

Example

```
SYSTEM sys;
WORD Result = HLL_QuerySystem(hWnd, &sys);
```

Function 21: Reset System

This function resets session parameters changed in Function 9, "Set Session Parameters," to their default state and releases any reserved sessions. This function also releases any connected PS, and cancels any keystroke interceptions and host update monitors.

An application can call this function at any time to restore session parameters to default values. This function should always be called just before a HLLAPI application program exits.

Syntax

```
WORD HLL_ResetHLLWin (HWND hWnd);
```

Prerequisites

None.

Applicable session parameters

None.

Call parameters

An application program must pass the following parameters when calling this function:

Function	HLL_ResetSystem
hWnd	Window handle of the application

Return parameters

Result code

Function returns one of the following codes:

HLL_SUCCESS	The function succeeded.
HLL_SYSTEMERROR	A system error code for the specified client application.

Example

```
WORD Result = HLL_ResetSystem(hWnd);
```

Function 22: Query Session Status

This function returns specific information about the specified session. It returns the following information in the data string:

- Short and long names
- Terminal type
- Characteristics (EAB and programmed symbol support)
- Session usage
- Number of rows and columns in the PS

This function provides more information on individual sessions than the allsessions call (Function 10, "Query Sessions").

Syntax

WORD HLL_QuerySessionStatus (HWND hWnd, LPSESSIONSTATUS lpSessionStatus);

Prerequisites

None.

Applicable session parameters

None.

Call parameters

An application program must pass the following parameters when calling this function:

Function	HLL_QuerySessionStatus
hWnd	Window handle of the application
lpSessionStatus	An 16-byte structure to receive session information, the first byte of which contains the session short name.

Return parameters

Result code

Function returns one of the following codes:

HLL_SUCCESS	The function succeeded.
HLL_INVALIDPSID	No session configured with short-name specified.

HLL_INVALIDPARAMETER	lpSessionStatus is null.
HLL_SYSTEMERROR	A system error occurred; the function failed.

Session information

If successful, function replaces content of call parameter lpSessionStatus with information about the session, organized as follows:

Byte	Description
1	Session short name
2-9	Session long name
10	Session type: 'A' = HP T27 terminal 'D' = 3270 terminal 'E' = 3270 printer 'F' = 5250 terminal 'G' = 5250 printer 'O' = UNISYS UTS60 terminal 'V' = UNIX or VAX
11	Session characteristics: Bit 0: 0=No EAB; 1=EABs Bit 1: 0=No programmed symbols 1=Programmed symbols Bit 2–7: Reserved
12	Usage state: logical OR of 10000000 = configured 01000000 = opened 00100000 = powered 00010000 = autopowered 00001000 = emulated (visible) 00000100 = HLLAPI-connected 00000010 = in file transfer
13–14	Number of rows (binary).
15–16	Number of columns (binary).

Example

```
SESSIONSTATUS ss;
ss.cPSID = 'J';
WORD Result = HLL_QuerySessionStatus(hWnd, &ss);
```

Function 23: Start Host Notification

This function begins the process by which HLLAPI determines if the host session PS or OIA has been updated. Your application can then call Function 24, "Query Host Update," to find out more specific information about the update. This function also enables the designated host session event to end an interruptible pause started with Function 18, "Pause."

If Windows Message Service notification is requested, an XM_SESSIONUPDATE message will be posted to the application when HLLAPI receives updates of the specified type. This avoids the application querying for updates. See Appendix E, "Attachmate HLLAPI Messages," for more information on Windows XM messages. Even though a client receives XM_SESSIONUPDATE messages, HLLAPI continues to track updates, so interruptible pauses and Function 24, "Query Host Update," operate normally.

Syntax

```
WORD HLL_StartHostNotification(HWND hWnd, char cPSID,
                               WORD wNotificationType, BOOL bWindowsMessage);
```

Prerequisites

None.

Applicable session parameters

None.

Call parameters

An application program must pass the following parameters when calling this function:

Function	HLL_StartHostNotification
hWnd	Window handle of the application
cPSID	The session short-name character.
wNotificationType	A word specifying the type(s) of notifications requested, a logical OR of the following: HLL_NOTIFYALLUPDATE HLL_NOTIFYCURSORUPDATE HLL_NOTIFYOIAUPDATE HLL_NOTIFYPSUPDATE HLL_NOTIFYBEEP
bWindowsMessage	If TRUE, indicates the application requests

XM_SESSIONUPDATE messages on updates.

Return parameters

Result code

Function returns one of the following codes:

HLL_SUCCESS	The function succeeded.
HLL_INVALIDPSID	Not a valid PSID.
HLL_INVALIDPARAMETER	An error was made in designating parameters.
HLL_INVALIDPSPOSITION	An invalid PS position was specified.
HLL_SYSTEMERROR	A system error occurred; the function failed.

Example

```
/* Request notification of OIA updates for session B */
WORD Result = HLL_StartHostNotification(hWnd, 'B', 2, 0);
```

Function 24: Query Host Update

This function allows your application to determine if the host has updated the PS or OIA since the last time Function 23, "Start Host Notification" or this function was called.

The application program need not be connected to the PS for updates; however, it must specify the short name for the desired session.

Syntax

DWORD HLL_QueryHostUpdate (HWND hWnd, char cPSID);

Prerequisites

Function 23, "Start Host Notification."

Applicable session parameters

None.

Call parameters

An application program must pass the following parameters when calling this function:

Function	HLL_QueryHostUpdate
hWnd	Window handle of the application
cPSID	A char containing the short name of the desired session

Return parameters

Function returns a double word (4 bytes).

Result code

The low-order word of the return value contains one of the following codes:

HLL_SUCCESS	The updates were received; see HIWORD.
HLL_INVALIDPSID	Invalid short-name character.
HLL_NOPRIORSTARTHOSTNOTIFY	Function 23, "Start Host Notification," has not been called.
HLL_SYSTEMERROR	A system error occurred; the function failed.
HLL_NOSESSIONUPDATE	There were no session updates since last call.

Updates posted

The high-order word of the return value indicates the type(s) of notifications received, a logical OR of the following:

HLL_NOTIFYCURSORUPDATE
HLL_NOTIFYOIAUPDATE
HLL_NOTIFYPSUPDATE
HLL_NOTIFYBEEP
HLL_NOTIFYBASECOLORCHANGE
HLL_NOTIFYMODELCHANGE
HLL_NOTIFYPOWERCHANGE

Example

```
DWORD Result = HLL_QueryHostUpdate(hWnd, 'B');
```

Function 25: Stop Host Notification

This function disables the capability of Function 24, "Query Host Update." This function can also be used to stop host events from affecting Function 18, "Pause."

Syntax

```
WORD HLL_StopHostNotification (HWND hWnd, char cPSID);
```

Prerequisites

Function 23, "Start Host Notification."

Applicable session parameters

None.

Call parameters

An application program must pass the following parameters when calling this function:

Function	HLL_StopHostNotification
hWnd	Window handle of the application
cPSID	A char containing the short name of the session for which notification is to be stopped.

Return parameters

Result code

Function returns one of the following codes:

HLL_SUCCESS	The function succeeded.
HLL_INVALIDPSID	Invalid session short-name supplied.
HLL_NOPRIORSTARTHOSTNOTIFY	Function 23, "Start Host Notification," has not been called.
HLL_SYSTEMERROR	A system error occurred; the function failed.

Example

```
WORD Result = HLL_StopHostNotification(hWnd, 'B');
```

Function 30: Search Field

This function searches through a specified field of the currently connected PS for a specified string. It can be used to search for a string in either protected or unprotected fields of a field formatted host PS. If the target string is found, this function returns the starting position of the string.

This search is always case-sensitive. This function requires a complete match of target string to field contents, regardless of the direction of the search.

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

Syntax

```
DWORD HLL_SearchField (HWND hWnd, LPSTR  lpSearchString,
                       WORD wStringLength, WORD wPSP);
```

Prerequisites

Function 1, "Connect Presentation Space."

Applicable session parameters

The following session parameters from Function 9 affect this function.

SRCHALL (default)

The entire field containing the specified PS position is searched.

SRCHFROM

Search begins at the specified position in the field.

SRCHFRWD (default)

Search finds first instance of the string between the origin and the end of the field.

SRCHBKWD

Search finds the last instance of the string between the field origin and the end of the field, or the specified PS position (if SRCHFROM is set).

Call parameters

An application program must pass the following parameters when calling this function:

Function	HLL_SearchField
hWnd	Window handle of the application
lpSearchString	ASCII text to be searched for in the field

wStringLength	Number of characters in lpSearchString to be used as search target.
wPSP	Specifies a PS position within the target field or on the field attribute that begins it.. For SRCHALL, this can be any PS position within the field. For SRCHFROM, search begins here for SRCHFRWD or ends here for SRCHBKWD.

Return parameters

Function returns a double word (4 bytes).

Result code

The low-order word of the return value contains one of the following codes:

HLL_SUCCESS	The function succeeded.
HLL_INVALIDPSID	Not connected
HLL_INVALIDPSPOSITION	An invalid PS position was specified.
HLL_SYSTEMERROR	A system error occurred; the function failed.
HLL_SEARCHSTRINGNOTFOUND	The search string was not found, or the screen was unformatted.

Location

The high-order word of the return value indicates the PS position where the specified text was found. If zero, the specified text was not found.

Example

```
/* Search field at PS position 199 for "Hello" */
DWORD Result = HLL_SearchField(hWnd, "Hello", 5, 199);
```

Function 31: Find Field Position

This function searches through the currently connected PS for a field's beginning position and returns the position. This function will search for either protected or unprotected fields, but the fields must be in a field-formatted host PS.

Syntax

```
DWORD HLL_FindFieldPosition(HWND hWnd, WORD wFieldSpecifier,
                            WORD wTargetPSP);
```

Prerequisites

Function 1, "Connect Presentation Space."

Applicable session parameters

None.

Call parameters

An application program must pass the following parameters when calling this function:

Function	HLL_FindFieldPosition
hWnd	Window handle of the application
wFieldSpecifier	This contains one of the following values from the header file: HLL_THISFIELD HLL_NEXTFIELD (protected or unprotected) HLL_NEXTPROTECTEDFIELD HLL_NEXTUNPROTECTEDFIELD HLL_PREVIOUSFIELD (protected or unprotected) HLL_PREVIOUSPROTECTEDFIELD HLL_PREVIOUSUNPROTECTEDFIELD
wTargetPSP	This contains the PS position where you want to start the search.

Return parameters

Function returns a double word (4 bytes).

Result code

The low-order word of the return value contains one of the following codes:

HLL_SUCCESS	The function succeeded. (See HIWORD.)
HLL_INVALIDPSID	Not connected.
HLL_INVALIDPSPOSITION	An invalid PS position was specified.

HLL_SYSTEMERROR	A system error occurred; the function failed.
HLL_NOSUCHFIELD	No such field was found, or the PS was unformatted.
HLL_INVALIDPARAMETER	An invalid option was specified.

Field position

The high-order word of the return value indicates the PS position where the specified field begins (one character past the field attribute byte in the presentation space.; = 1 if the field attribute is in the last position of the PS).

Example

```
/* Find start of the field that includes PS position 199*/
DWORD Result = HLL_FindFieldPosition(hWnd, HLL_THISFIELD, 199);
```

Function 32: Find Field Length

This function returns the length of a specified PS field, protected or unprotected, and is the number of characters contained in the field between the attribute byte that begins the field and the next-following field attribute.

NOTE. This function wraps from the end to the beginning of the PS.

Syntax

```
DWORD HLL_FindFieldLength(HWND hWnd, WORD wFieldSpecifier,
                          WORD wTargetPSP);
```

Prerequisites

Function 1, "Connect Presentation Space."

Applicable session parameters

None.

Call parameters

An application program must pass the following parameters when calling this function:

Function	HLL_FindFieldLength
hWnd	Window handle of the application
wFieldSpecifier	This contains one of the following values from the header file: HLL_THISFIELD HLL_NEXTFIELD (protected or unprotected) HLL_NEXTPROTECTEDFIELD HLL_NEXTUNPROTECTEDFIELD HLL_PREVIOUSFIELD (protected or unprotected) HLL_PREVIOUSPROTECTEDFIELD HLL_PREVIOUSUNPROTECTEDFIELD
wTargetPSP	This contains the PS position where you want to start the search.

Return parameters

Function returns a double word (4 bytes).

Result code

The low-order word of the return value contains one of the following codes:

HLL_SUCCESS	The function succeeded. (See HIWORD.)
HLL_INVALIDPSID	Not connected.
HLL_INVALIDPSPOSITION	An invalid PS position was specified.
HLL_SYSTEMERROR	A system error occurred; the function failed.

HLL_NOSUCHFIELD	No such field was found, or the PS was unformatted.
HLL_ZEROLENTHFIELD	The field length is zero.
HLL_INVALIDPARAMETER	The value of the given wFieldSpecifer parameter was not within the range this function requires.

Field position

The high-order word of the return value indicates the length of the field specified. . If zero, the field was not found, or is zero length, or the PS is unformatted.

Note If a field attribute is followed by another field attribute, the field is assumed to have a length of zero.

Example

```
/* Find length of the field that includes PS position 199*/
DWORD Result = HLL_FindFieldLength(hWnd, HLL_THISFIELD, 199);
```

Function 33: Copy String to Field

This function copies characters to a specific unprotected field in a field-formatted PS.

The copy operation ends when one of four conditions is met:

- The entire string has been copied.
- The text has been written to the last field position.
- The function has copied the specified number of characters in the data length parameter.
- The character before the EOT character is copied when EOT is specified.

Note AID key character sequences are not evaluated when using this function, but are copied to the field as literal strings. Function 3, "Send Key," must be used to send an AID key to a session.

Syntax

```
WORD HLL_CopyStringToField(HWND hWnd, LPSTR lpString, WORD wStringLength,
                           WORD wFieldPSP);
```

Prerequisites

Function 1, "Connect Presentation Space."

Applicable session parameters

The following session parameters from Function 9 affect this function.

EAB

Text and EABs are copied from the data string.

NOEAB (default)

The data string does not contain EABs.

Call parameters

An application program must pass the following parameters when calling this function:

Function	HLL_CopyStringToField
hWnd	Window handle of the application
lpString	Text and, if session parameter EAB is set, EABs to be copied into the field. Refer to Appendix D, "Extended Attributes," for information on EAB formats.
wStringLength	Number of characters to be copied from Data string.

PS position	A position in the PS that lies within the field or on the field attribute that begins it. Copy always starts at the beginning of the field.

Return parameters

Result code

Function returns one of the following codes:

HLL_SUCCESS	The function succeeded.
HLL_INVALIDPSID	Not connected.
HLL_INVALIDPARAMETER	An invalid parameter string (for example, lpString null or wStringlength=0) wasspecified.
HLL_PROTECTED	The target was protected or inhibited, or illegal data was sent to the target field (for example, a field attribute byte).
HLL_TRUNCATED	The copy was completed, but data may have been truncated.
HLL_INVALIDPSPOSITION	An invalid PS position was specified.
HLL_SYSTEMERROR	A system error occurred; the function failed.
HLL_UNFORMATTEDHOSTPS	The host PS was unformatted.

Example

```
/* Copy "Hello World" to field containing position 199 */
WORD Result = HLL_CopyStringToField(hWnd, "Hello World", 11, 199);
```

Function 34: Copy Field to String

This function copies all characters from a field in the currently connected PS into a string. It can be used with either protected or unprotected fields, but only in a field-formatted PS.

The copy operation begins at the field's origin. This position and length information can be found by using Function 31, "Find Field Position," and Function 32, "Find Field Length."

This function ends when one of two conditions is met:

- The last character in the field was copied.
- All character positions in the copy string have been filled.

Syntax

```
WORD HLL_CopyFieldToString(HWND hWnd, LPSTR lpBuffer,
                           WORD wBufferLength, WORD wPSP);
```

Prerequisites

Function 1, "Connect Presentation Space."

Applicable session parameters

The following session parameters from Function 9 affect this function.

EAB

Text and EABs are copied to the buffer.

NOEAB (default)

EABs are not copied.

Call parameters

An application program must pass the following parameters when calling this function:

Function	HLL_CopyFieldToString
hWnd	Window handle of the application
lpBuffer	The string to which the program copies the contents of the field.
wBufferLength	The number of characters to be copied, allowing for EABs if session parameter EAB is set.
WPSP	A position in the PS that lies within the field or on the field attribute that begins it. Copy starts at the beginning of the field.

Return parameters

Field content

Function replaces content of call parameter lpBuffer with text and, if requested, EABs from the field.

Refer to Appendix D, "Extended Attributes," for information on EAB interpretation.

Result code

Function returns one of the following codes:

HLL_SUCCESS	The function succeeded.
HLL_INVALIDPSID	Not connected.
HLL_INVALIDPARAMETER	An invalid parameter string (for example, lpString null or wStringlength=0) was specified.
HLL_FIELDSIZEMISMATCH	The data to be copied and the target field were not the same size.
HLL_INVALIDPSPOSITION	An invalid PS position was specified.
HLL_SYSTEMERROR	A system error occurred; the function failed.
HLL_UNFORMATTEDPS	The host PS was unformatted.

Example

```
/* Allocated data buffer */
char DataStr[11];
/* Copy ten characters, w/o EABs, from field at PS pos 199 */
WORD Result = HLL_CopyFieldToString(hWnd, DataStr, 10, 199);
```

Function 40: Set Cursor

This function sets the cursor position to the target PS position in the currently connected PS.

There is no direct way to move the cursor in Control Unit Terminal (CUT) mode. This function can be simulated by using keystrokes.

Syntax

WORD HLL_SetCursor (HWND hWnd, WORD wCursorLocation);

Prerequisites

Function 1, "Connect Presentation Space."

Applicable session parameters

None.

Call parameters

An application program must pass the following parameters when calling this function:

Function	HLL_SetCursor
hWnd	Window handle of the application
wCursorLocation	The desired cursor position in the PS. (Two bytes)

Return parameters

Result code

Function returns one of the following codes:

HLL_SUCCESS	The function succeeded.
HLL_INVALIDPSID	Not connected.
HLL_SESSIONOCCUPIED	The session is busy.
HLL_INVALIDPSPOSITION	An invalid PS position was specified (the cursor position is less than 1 or greater than PS maximum.)
HLL_SYSTEMERROR	A system error occurred; the function failed.

Example

```
/* Put cursor in PS position 199 */
WORD Result = HLL_SetCursor(hWnd, 199);
```

Function 50: Start Keystroke Intercept

This function allows an 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 buffer overflows or call is made to Function 21, "Reset System," or Function 53, "Stop Keystroke Intercept."

Intercepted keystrokes can be

- received through Function 51, "Get Key," and sent to the same or another session with Function 3, "Send Key"
- accepted or rejected through Function 52, "Post Intercept Status"
- replaced by other keystrokes with Function 3, "Send Key"
- used to trigger other processes.

As an alternative to using the delay version of "Get Key," an application can also request that Windows message, XM_KEYSTROKEINTERCEPTED, be posted whenever a keystroke pf the specified type is received.

Note: Extended processing of each keystroke may cause unacceptable delays for keyboard users.

If AID-key-only intercept is requested (option "D" is specified), non-AID keys are sent to the PS; only AID keys will be available to the application.

An application may intercept keystrokes for more than one session at a time, but can send keystrokes only to the currently connected session.

Syntax

```
WORD HLL_StartKeystrokeIntercept (HWND hWnd,
                                  LPSTARTINTERCEPT lpIntercept);
```

Prerequisites

None.

Applicable session parameters

None.

Call parameters

An application program must pass the following parameters when calling this function:

Function	HLL_StartKeystrokeIntercept
hWnd	Window handle of the application

Intercept spec A 6-byte structure specifying the intercept desired. (See format below.)

Intercept spec format

Byte	Description
1	Session short name character.
2-3	WORD - Intercept type. One of the following values: HLL_INTERCEPTAIDKEYS HLL_INTERCEPTALLKEYS
4-5	WORD Length of HLLAPI keystroke queue. Each keystroke requires 2 bytes of storage (a queue of 128 bytes holds 64 keystrokes).
6	If TRUE, HLLAPI posts the XM_KEYSTROKEINTERCEPTED message when a keystroke is intercepted (see **Appendix E, "Attachmate HLLAPI Messages,"** for information on XM messages). If FALSE, no message is posted.

Return parameters

Result code

Function returns one of the following codes:

HLL_SUCCESS	The function succeeded.
HLL_INVALIDPSID	Invalid session short name supplied.
HLL_INVALIDPARAMETER	An invalid parameter string (for example, lpString null or wStringlength=0) was specified.
HLL_RESOURCEUNAVAILABLE	The function failed; another client application window has a keystroke interceptor on the target session.
HLL_SYSTEMERROR	A system error occurred; the function failed.
HLL_MEMORYNOTAVAILABLE	The queue can't be allocated.
HLL_NOEMULATORATTACHED	This call requires an emulator attached to the session.

Example

```
/* Start intercept of AID keystrokes in session B */
STARTINTERCEPT IntrcptSpec;
IntrcptSpec.cPSID = 'B' ;
IntrcptSpec.wKeyFilter = HLL_INTERCEPTAIDKEYS;
IntrcptSpec.wQueueLength = 127;
IntrcptSpec.bWindowsMessage = TRUE;
WORD Result = HLL_StartKeystrokeIntercept(hWnd, &IntrcptSpec);
```

Notes

Specifying the bWindowsMessage as TRUE is an excellent way of avoiding a Function 51, "Get Key," delay. The client application can trap the XM_KEYSTROKEINTERCEPTED message and call Function 51, "Get Key," only when it receives the message.

Extended processing of each keystroke by setting HLL_INTERCEPTALLKEYS may cause unacceptable delays for keyboard users.

Function 51: Get Key

This function allows your application to receive the keystrokes for the sessions that were specified with Function 50, "Start Keystroke Intercept."

Use Function 3, "Send Key," to pass keystrokes to the target PS.

When keystrokes are available, they are read into the data area that you have provided in your client application program. Each keystroke is represented by one of the key codes listed in Appendix B, "Keyboard Mnemonics."

The CAPSLOCK key on the PC works like the SHIFTLOCK key on the host system; it produces the uppercase of all keys, not just alphanumeric keys. So if the application is getting keys with CAPSLOCK on, it gets all keys in the shifted state.

HLLAPI can also notify your application with a message when a message is received. Then the function will fetch the keystroke with no delay, whatever Wait parameter value is specified. See Function 50, "Start KeystrokeIntercept."

Syntax

```
WORD HLL_GetKey (HWND hWnd, LPKEYSTROKE lpKeystroke);
```

Prerequisites

Function 50, "Start Keystroke Intercept."

Applicable session parameters

The following session parameters from Function 9 affect this function.

TWAIT (default)

The function does not return control to the calling application until a key has been intercepted.

LWAIT

The function does not return control to the calling application until a key has been intercepted.

NWAIT

The function checks for intercepted keystrokes and returns immediately. NWAIT means that Function 51, "Get Key, will return the code HLL_KEYSTROKESNOTAVAILABLE if nothing is queued that matches the option specified under Function 50, "Start Keystroke Intercept."

Call parameters

An application program must pass the following parameters when calling this function:

Function	HLL_GetKey
hWnd	Window handle of the application
lpKeystroke	An 8-character code specifying the intercept desired. (See format below.)

Data string format

Byte	Description
1	Session short name character. If blank or null, the session to which the application is currently connected.
2-8	Blanks reserving space for the intercepted data.

Return parameters

Intercept string

Function replaces content of call parameter lpKeystroke with information describing the keystroke intercepted.

Byte	Description
1	A 1-character session short name; if or blank/null indicating a intercept is for the currently connected PS.
2	One of : A= ASCII returned, or M= Keystroke mnemonic
3-7	Allocated buffer used for queuing and dequeuing keystrokes. This buffer contains the following: • If the key returned is a character key, bytes 3 and 4 contain the ASCII character followed by 00. If it is a 3270 key, bytes 3 and 4 will contain a mnemonic for the keystroke (for example, @5 represents PF5). • Bytes 5 through 7 contain nulls, unless the key returned was a combination key such as ERASEINPUT, for which bytes 3 through 6 would contain @A@F and bytes 7-8 nulls.

Typical intercept strings

Intercept strings Function 51 might return are shown below with their keyboard equivalents

Intercept string	Keyboard equivalent
Eat	"E" represents the session and "A" informs your HLLAPI application that the keystrokes will be sent as ASCII; the returning key is a lowercase T (Bytes 4–7 = X'00').
EM@2	"E" represents the session, "M"indicates that the keystrokes will be returned as key mnemonics, and "@2" indicates the key being returned is PF2 (Bytes 5–7 = X'00').
ES@A@2	"E" represents the session, "S" indicates that the keystrokes will be returned in a special shift state, and "@A@2" indicates the key being returned is ALT PF2 (Byte 7 = X'00').
ES@r@2	"E" represents the session, "S" indicates that the keystrokes will be returned in a special shift state, and "@r@2" indicates the key being

	received is CTRL PF2 (Byte 7 = X'00').
EM@E	"E" represents the session, "M" indicates the keystrokes will be returned as mnemonics, and "@E" indicates that the key being received is ENTER (Bytes 5–7 = X'00').

Result code

Function returns one of the following codes:

HLL_SUCCESS	The function succeeded.
HLL_INVALIDPSID	Invalid session short-name supplied.
HLL_PSLOCKED	An attempt to enter a non-AID keystroke was made after the client application specified the HLL_INTERCEPTAIDKEY intercept code in Function 50, "Start Keystroke Intercept."
HLL_NOPRIORSTARTKEYSTROKE	No prior call to Function 50, "Start Keystroke Intercept," was made for this PS.
HLL_SYSTEMERROR	A system error occurred; the function failed.
HLL_KEYSTROKENOTAVAILABLE	The requested keystrokes are not available on the input queue.
HLL_KEYSTROKEQUEUEOVERFLOW	The keystroke queue overflowed; keystrokes were lost.

Example

```
/* Allocate space for returning keystroke from session B */
KEYSTROKE k;
k.cPSID = 'B';
WORD Result = HLL_GetKey(hWnd, &k);
```

Function 52: Post Intercept Status

This function places a sentinel on keyboard input that sounds a beep if the keystroke obtained through Function 51, "Get Key," was rejected.

Syntax

WORD HLL_PostInterceptStatus (HWND hWnd, char cPSID, WORD wStatus);

Prerequisites

Function 50, "Start Keystroke Intercept."

Applicable session parameters

None.

Call parameters

An application program must pass the following parameters when calling this function:

Function	HLL_PostInterceptStatus
hWnd	Window handle of the application
cPSID	A string specifying the session short name.
WStatus	API_INTERCEPTACCEPTED or API_INTERCEPTREJECTED

Return parameters

Result code

Function returns one of the following codes:

HLL_SUCCESS	The function succeeded.
HLL_INVALIDPSID	Invalid session short-name supplied.
HLL_NOPRIORSTARTKEYSTROKE	No prior call to Function 50, "Start Keystroke Intercept," was made for this PS.
HLL_SYSTEMERROR	A system error occurred; the function failed.

Example

```
/* Post 'accepted' status to session B */
WORD Result = HLL_PostInterceptStatus(hWnd, 'B',
                                      API_INTERCEPTACCEPTED);
```

Function 53: Stop Keystroke Intercept

This function ends an application's ability to intercept keystrokes for the specified session.

Syntax

```
WORD HLL_StopKeystrokeIntercept (HWND hWnd, char cPSID);
```

Prerequisites

Function 50, "Start Keystroke Intercept."

Applicable session parameters

None.

Call parameters

An application program must pass the following parameters when calling this function:

Function	HLL_StopKeystrokeIntercept
hWnd	Window handle of the application
cPSID	The session short-name character.

Return parameters

Result code

Function returns one of the following codes:

HLL_SUCCESS	The function succeeded.
HLL_INVALIDPSID	Invalid session short name supplied.
HLL_NOPRIORSTARTKEYSTROKE	No prior Function 50, "Start Keystroke Intercept," was issued for this PS.
HLL_SYSTEMERROR	A system error occurred; the function failed.

Example

```
WORD Result = HLL_StopKeystrokeIntercept(hWnd, 'B');
```

Function 90: Send File

This function allows the client application program to send a file to a host session.

HLLAPI-initiated file transfers are synchronous, returning control on completion of the file transfer.

The program requesting synchronous file transfers must not be intercepting keystrokes for any sessions, must not be awaiting the outcome of another synchronous file transfer, and must not be waiting for host events in any session.

For asynchronous file transfers, an XM_FILETRANSFERCOMPLETION message is posted to the application upon completion. See Appendix E, "Attachmate HLLAPI Messages," for more information.

Syntax

```
DWORD HLL_SendFile (HWND hWnd, LPSTR lpszSendCommand,
                    LPWORD lpwSequenceID);
```

Prerequisites

The session to be used for a file transfer must be logged on and at a host system prompt.

Applicable session parameters

The following session parameters from Function 9 affect this function.

TWAIT = 1 (default)

File transfer is synchronous (function returns when the transfer completes).

LWAIT = 2

File transfer is synchronous (function returns when the transfer completes).

NWAIT = 3

File transfer is asynchronous (function returns immediately. WM_FILETRANSFERCOMPLETION message is posted when the transfer completes).

Call parameters

An application program must pass the following parameters when calling this function:

Function	HLL_SendFile
hWnd	Window handle of the application
lpszSendCommand	A string (maximum 128 bytes) containing the send command string.

lpwSequenceID	The far pointer to WORD (asynchronous only).

Return parameters

Function returns a double word (4 bytes).

Result code

The low-order word of the return value contains one of the following codes:

HLL_SUCCESS	The function succeeded.
HLL_INVALIDPSID	An invalid session short name was specified in the commandstring.
HLL_INVALIDPARAMETER	The command line is not valid (for example, command string is longer than 128 bytes).
HLL_SYSTEMERROR	A system error occurred; the function failed.
HLL_RESOURCEUNAVAILABLE	The session is already in use by file transfer.
HLL_TIMEOUT	The client application window time limit expired; the transfer was ended.
HLL_DELAYENDEDBYCLIENT	The application broke synchronous delay with a Wait, Pause or GetKey

Transfer outcome

For asynchronous transfer, the high-order word receives the handle to the file transfer application

For synchronous transfer,.the high-order word of the return value contains one of the following codes:

Code	Description
3	The file was transferred.
4	The file was transferred with records segmented.
5	Workstation file name not valid or file not found.
9	A system error occurred.
27	The file transfer was terminated by CTRL C.
301	Invalid function number.
302	File not found.
303	Path not found.
305	Access denied.
308	Insufficient memory.
310	Invalid environment.
311	Invalid format.

Return codes from the file transfer applications are returned to the client application window that requested synchronous file transfers as the HIWORD of the function DWORD. Asynchronous file transfers receive a file transfer application handle that identifies the file transfer application and the 16-bit sequence number. On completion of the transfer, the file transfer will receive an XM_FILETRANSFERCOMPLETION Windows message with the corresponding sequence number and the same file transfer completion code. See Appendix E, "Attachmate HLLAPI Messages," for more information on Windows messages.

Example

```
/* Send command string   Assumes        */
/*    PC filename = pcfile.ext          */
/*    Session short name = D            */
/*    Host filename = hostfile.ext      */
/*    CMS transfer options = ASCII,CRLF */
char HllDataStr [] = "pcfile.ext d: hostfile ext (ASCII CRLF";
WORD Result = HLL_SendFile(hWnd, HllDataStr, NULL);
```

Function 91: Receive File

This function allows the client application program to receive a file from a host session.

HLLAPI-initiated file transfers are synchronous, returning control on completion of the file transfer.

The program requesting synchronous file transfers must not be intercepting keystrokes for any sessions, must not be awaiting the outcome of another synchronous file transfer, and must not be waiting for host events in any session.

For asynchronous file transfers, an XM_FILETRANSFERCOMPLETION message is posted to the application upon completion. See Appendix E, "Attachmate HLLAPI Messages," for more information.

Syntax

```
DWORD HLL_SendFile (HWND hWnd, LPSTR lpszSendCommand,
                                LPWORD lpwSequenceID);
```

Prerequisites

The session to be used for a file transfer must be logged on and at a host system prompt.

Applicable session parameters

The following session parameters from Function 9 affect this function.

TWAIT = 1 (default)

File transfer is synchronous (function returns when the transfer completes).

LWAIT = 2

File transfer is synchronous (function returns when the transfer completes).

NWAIT = 3

File transfer is asynchronous (function returns immediately. WM_FILETRANSFERCOMPLETION message is posted when the transfer completes).

Call parameters

An application program must pass the following parameters when calling this function:

Function	HLL_ReceiveFile
hWnd	Window handle of the application
lpszSendCommand	A string (maximum 128 bytes) containing the send command string.

lpwSequenceID	The far pointer to WORD (asynchronous only).

Return parameters

Result code

The low-order word of the return value contains one of the following codes:

HLL_SUCCESS	The function succeeded.
HLL_INVALIDPSID	An invalid session short name was specified in the commandstring.
HLL_INVALIDPARAMETER	The command line is not valid (for example, command string is longer than 128 bytes).
HLL_SYSTEMERROR	A system error occurred; the function failed.
HLL_RESOURCEUNAVAILABLE	The session is already in use by file transfer.
HLL_TIMEOUT	The client application window time limit expired; the transfer was ended.
HLL_DELAYENDEDBYCLIENT	The application broke synchronous delay with a Wait, Pause or GetKey

Transfer outcome

For asynchronous transfer, the high-order word receives the handle to the file transfer application

For synchronous transfer,.the high-order word of the return value contains one of the following codes:

Code	Description
3	The file was transferred.
4	The file was transferred with records segmented.
5	Workstation file name not valid or file not found.
9	A system error occurred.
27	The file transfer was terminated by CTRL C.
301	Invalid function number.
302	File not found.
303	Path not found.
305	Access denied.
308	Insufficient memory.
310	Invalid environment.
311	Invalid format.

Return codes from the file transfer applications are returned to the client application window that requested synchronous file transfers as the HIWORD of the function DWORD. Asynchronous file transfers receive a file transfer application handle that identifies the file transfer application and the 16-bit sequence number. On completion of the transfer, the file transfer will receive an XM_FILETRANSFERCOMPLETION Windows message with the corresponding sequence number and the same file transfer completion code. See Appendix E, "Attachmate HLLAPI Messages," for more information on Windows messages.

Example

```
/* Receive command string   Assumes     */
/*    PC filename = pcfile.ext          */
/*    Session short name = B            */
```

```
/*    Host filename = hostfile.ext      */
/*    CMS transfer options = ASCII,CRLF */
char HllDataStr [] = "pcfile.ext b: hostfile ext (ASCII CRLF";
DWORD Result = HLL_ReceiveFile(hWnd, HllDataStr, NULL);
```

Function 99: Convert Position or RowCol

This function converts a PS position value into display row/column coordinates or a row/column value into PS position display coordinates.

When the conversion is made, the function considers the model number of the host system display type being emulated. This function does not change the cursor position.

Syntax

```
WORD HLL_Convert(HWND hWnd, char cPSID,
                 WORD wPositionOrRowColumn, LPPOINT lpPoint);
```

Prerequisites

None.

Applicable session parameters

None.

Call parameters

An application program must pass the following parameters when calling this function:

Function	HLL_Convert
hWnd	Window handle of the application
cPSID	This specifies the target PS.
wPositionOrRowColumn	This contains one of the following identifiers from the include file: HLL_CONVERTPOSITION HLL_CONVERTROWCOLUMN
lpPoint	This points to a data structure that contains initial values. If wOption is HLL_CONVERTPOSITION, then lpPoint.x=wPSP to be converted. If wOption is HLL_CONVERTROWCOLUMN, then lpPoint.y=row and lpPoint.x=column.

Return parameters

lpPoint LPPOINT	This points to the data structure, which returns the converted values. The structure contains the row number in y and the column number in x, if wOption is HLL_CONVERTPOSITION. The structure contains a PS position in x if wOption is

HLL_CONVERTROWCOLUMN.

Result code

Function returns one of the following codes:

HLL_SUCCESS	The function succeeded.
HLL_INVALIDPSID	An invalid session short name was specified in the command string.
HLL_INVALIDPARAMETER	The option is other than HLL_CONVERTROWCOLUMN or HLL_CONVERTPOSITION, or null value for lpPoint.
HLL_INVALIDPSPOSITION	An invalid position or row/column was specified.
HLL_SYSTEMERROR	A system error occurred; the function failed.

Example

```
POINT RowCol;
/* Convert position 199 of session B to row column */
RowCol.x = 199;
WORD HLL_Convert(hWnd, "B", 2, &RowCol);
```

Appendix A: General troubleshooting procedures

If you have problems running your automation software with Attachmate product, consider the following.

1. **Check that EXTRA! is in the path**. Often the reason an application will fail to start is that the system cannot find the emulator software. At a command prompt, type EXTRA and press Enter. A response like "Unknown command or file name" indicates EXTRA! is not in the system search path. Make needed correction, then re-test to verify.

2. **Check the configuration options**. Many problems occur when a session with a short name required by an application has not been configured. Start a session, choose Global Preferences… from the Options menu, then select Advanced properties. Verify that the HLLAPI short name needed by the application has an appropriate session assigned. If not, make needed correction and run the application again to verify.

3. **Check connections**. While faulty cable connections are rare in newer hardware, inspect plugs and jacks to confirm they are securely attached. A more common cause of "failed to connect" errors is improper specification of connection parameters, for example, host TCP/IP network address. Use a technique such as PING to check the connection configuration, and correct as necessary.

4. **Check the session**. On occasion, host application programmers may modify content or organization of screens to meet changing need. If workstation automation software has been written to expect specific text in a particular place on a particular screen, software error of some kind is likely to result. Because host applications are rarely changed without notice, systematically review all such advisories. In the event an issue of this type does occur, use a tool such as an API trace to determine exactly where in the software failure occurs, then use that information to identify specifics of the change, and develop appropriate updates for automation software.

5. **Check workload and timings**. If an automation program has been in use for several years, chances are good that hardware at the host, in the network, or the workstation will have been upgraded – or, if not, that workloads on the hardware have changed. In either case, time required to receive and process requests will change, possibly enough that host applications and automation software can get "out of synch", expecting (and trying to process) information that has not yet arrived. Problems like these can be perplexing to diagnose and resolve. Review automation-software logic to verify that suitably robust techniques are being used to synchronize host

and workstation operations. If necessary, Attachmate Technical Support can assist by analyzing communications traces to provide information about turnaround times and other details of host/workstation data exchanges.

Appendix B: Host keyboard mnemonics

Table B-1 shows the key codes that allow you to represent special function keys in your calling data strings. You can use these codes with Function 3, "Send Key," to specify the keystrokes you want to send, as well as with Function 51, "Get Key," which receives the keystrokes sent through Function 3.

These codes rely on ASCII characters to represent the special function keys of the 3270-PC. For example, to send the keystroke PF1, you would code "@1". And to represent a System Request keystroke, you would code "@A@H".

Each key code represents the actual key that is being sent or received. Keep in mind that placing an Alt (@A) or Shift (@S) before a key code will change its meaning. When sending text keystrokes, be sure the codes are entered just as you want them to be received, including the correct case.

Since the Escape character defaults to the at sign (@), you must code the character twice in order to send the escape character as a keystroke. For example, to send a single "@", you must code "@@". When your program calls Function 51, "Get Key," you send a pointer to a

keystroke structure used for the returning keystroke. Each keystroke is represented by the following key codes:

- Each key has a number between 1 and 133, which represents the key position on the keyboard.
- Every key has four states: Lower Case, Upper Case, Alt State, and Ctrl State.

Symbols used throughout the tables have the following meanings:

#	Shift keys: this symbol indicates that what follows will be a mnemonic key code.
*	These key positions are not used.
E	A host session's short name.

Table B-1. Windows keyboard mnemonics

Host key	Mnemonic	Host key	Mnemonic
@	@@	Home	@0
Alternate Cursor	@$	Insert	@I
Attention	@A@Q	Jump	@J
Backspace	@<	New Line	@N
Backtab	@B	Num Lock	@t
Blue	@A@h	Page Down	@v
Caps Lock	@Y	Page Up	@u
Clear	@C	PA1	@x
Cursor Down	@V	PA2	@y
Cursor Left	@L	PA3	@z
Cursor Left Double	@A@L	PF1	@1
Cursor Right	@Z	PF2	@2
Cursor Right Double	@A@Z	PF3	@3
Cursor Select	@A@J	PF4	@4
Cursor Up	@U	PF5	@5
Delete	@D	PF6	@6
Delete Word	@A@D	PF7	@7
Device Cancel	@A@R	PF8	@8
DUP	@S@x	PF9	@9
End	@q	PF10	@a
Enter	@E	PF11	@b
Erase to EOF	@F	PF12	@c
Erase Input	@A@F	PF13	@d
Reset Reverse Video	@A@c	PF14	@e
Field Mark	@S@y	PF15	@f
Green	@A@f	PF16	@g
Reset Host Colors	@A@l	PF17	@h
Reverse Video On	@A@9	PF18	@i
Scr Lock	@s	PF19	@j
System Request	@A@H	PF20	@k
Tab	@T	PF21	@l
Test	@A@C	PF22	@m
Turquoise	@A@i	PF23	@n
Underscore	@A@b	PF24	@o
White	@A@j	Pink	@A@e
Word Tab Back	@A@z	Print PS	@A@T
Word Tab Forward	@A@y	Print Screen	@P
Yellow	@A@g	Queue Overrun	@/
(reserved)	@X	Red	@A@d
Reset	@R	Field Exit	@A@E
Cursor Up Double	@A@U	Cursor Down Double	@A@V

Appendix C: Interpreting the Returned Data String for Function 13

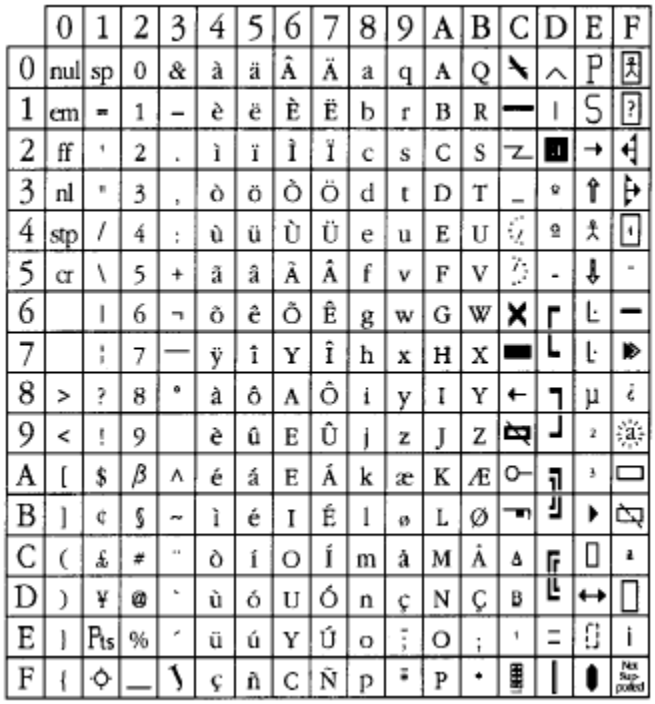
This appendix explains how to decode the data string that Function 13, "Copy OIA," returns. To interpret this information, you must be able to decipher the OIA image symbols that are returned in positions 2 to 81 of the string, as well as the bits that are returned in positions 82 to 103 of the string.

Position 1 (OIA format byte)

Position 1 of the returning data string always returns the format byte, 1 for 3270 terminal emulation or 9 for 5250.

Positions 2 to 81 (OIA image symbols)

The following chart displays symbols found in the DFT host and CUT host presentation spaces. These symbols can be part of the OIA image returned in positions 2 to 81 of the returning data string.



Positions 82 to 103 (OIA bit groups)

Remaining positions in the returned data string can be interpreted with the help of the following sections. Each position or group returns a bit number that explains a particular OIA characteristic. The list below summarizes the different groups, the OIA characteristic, and the position number associated with it.

Group	Characteristic explained	Position number
1	Online and Screen Ownership	82
2	Character Selection	83
3	Shift State	84
4	PSS, Part 1	85
5	Highlight, Part 1	86
6	Color, Part 1	87
7	Insert	88
8	Input Inhibited (5 bytes)	89–93
9	PSS, Part 2	94
10	Highlight, Part 2	95
11	Color, Part 2	96
12	Communication Error Reminder	97
13	Printer Status	98
14	Reserved (3270) / Graphic (5250)	99
15	Reserved Group	100
16	Automatic Key Play/Record Status	101
17	Automatic Key Quit/Stop State	102
18	Enlarge State Position	103

Group1: Online and screen ownership

This bit group is the 82nd byte of the OIA data returned to an application by Function 13. This group contains 1 byte of information, describing who owns the current session.

Bit	3270 Description	5250 Description
0	Setup	Reserved
1	Test	Reserved
2	SSCP–LU session owns screen	Reserved
3	LU–LU session owns screen	System available
4	Online and not owned	Reserved
5	Subsystem ready	Subsystem ready
6–7	Reserved	Reserved

Group 2: Character selection

This group is the 83rd byte in the OIA data returned to an application by Function 13. The group contains 1 byte of data and defines the character set currently used in the OIA.

Bit	3270 Description	5250 Description
0	Extended select	Reserved
1	APL	Reserved
2	Kana	Katakana (Japan only)
3	Alphanumeric	Alphanumeric
4	Text	Reserved
5	Reserved	Reserved
6	Reserved	Hiragana (Japan only)
7	Reserved	Double-byte character

Group 3: Shift state

This group is the 84th byte in the OIA data, showing whether caps lock and numeric lock are active.

Bit	3270 Description	5250 Description
0	Upper Shift	Reserved
1	Numeric	Keyboard shift
2	CAPS	CAPS
3-6	Reserved	
7	Reserved	Double-byte char available

Group 4: Program symbol support, part 1

This group is the 85th byte in the OIA data.

Bit	Description
0–7	Reserved

Group 5: Highlight, part 1

This group is the 86th byte in the OIA data and contains highlighting information for the current PS.

Bit	3270 Description	5250 Description
0	User selectable	Reserved
1	Field inherit	Reserved
2–7	Reserved	Reserved

Group 6: Color, part 1

This group is the 87th byte in the OIA data, defining some of the color characteristics being used in the current PS by this operator.

Bit	3270 Description	5250 Description
0	User selectable	Reserved
1	Field inherit	Reserved
2–7	Reserved	Reserved

Group 7: Insert

This group is the 88th byte in the OIA data, defining whether the current PS is in insert mode.

Bit	Description
0	Insert mode
1–7	Reserved

Group 8: Input inhibited

This group consists of bytes 89 through 93 in the OIA data, and indicates why input is inhibited in the current PS.

Byte	Bit	3270 Description	5250 Description
1	0	Non-resettable machine check	Reserved
	1	Reserved for security key	Reserved
	2	Machine check	Reserved
	3	Communications check	Reserved
	4	Program check	Reserved
	5-7	Reserved	Reserved
2	0	Device busy	Reserved
	1	Terminal wait	Reserved
	2	Minus symbol	Reserved
	3	Minus function	Reserved
	4	Too much entered	Reserved
	5-7	Reserved	Reserved
3	0-2	Reserved	Reserved
	3	Invalid dead key combination	Reserved
	4	Wrong place	Reserved
	5	Reserved	Operator input error
	6-7	Reserved	Reserved
4	0-1	Reserved	Reserved
	2	System wait	System wait
	3-7	Reserved	Reserved
5	0-7	Reserved	Reserved

Group 9: Program symbol support, part 2

This is the 94th byte of the OIA data, providing additional information about program symbol support.

Bit	Description
0–7	Reserved

Group 10: Highlight, part 2

This is the 95th byte in the OIA data, and defines more highlight options in the current PS.

Bit	Description
0–7	Reserved

Group 11: Color, part 2

This is the 96th byte in the OIA data. The group defines more color options available to the operator in the information area.

Bit	Description
0–7	Reserved

Group 12: Communications error reminder

This is the 97th byte in the OIA data. Bits in this group define whether the host and the current PS are communicating.

Bit	3270 Description	5250 Description
0	Communications error	Reserved
1–6	Reserved	Reserved
7	Reserved	Message wait

Group 13: Printer status error reminder

This is the 98th byte in the OIA data. Bits in this group describe the status of the printer connected to the current PS.

Bit	Description
0–7	Reserved

Group 14: Reserved (3270) / Graphics (5250)

This is the 99th byte in the OIA data.

Bit	Description
0–7	Reserved

Group 15: Reserved

This is the 100th byte in the OIA data.

Bit	Description
0–7	Reserved

Group 16: Automatic key play/record state

This group is the 101st byte in the OIA data.

Bit	Description
0–7	Reserved

Group 17: Automatic key quit/stop state

This group is the 102nd byte in the OIA data.

Bit	Description
0–7	Reserved

Group 18: Expanded state

This is the 103rd byte in the OIA data.

Bit	Description
0–7	Reserved

Appendix D: Extended Attributes

Function 5, "Copy Presentation Space," Function 8, "Copy Presentation Space to String," Function 15, "Copy String to Presentation Space," Function 33, "Copy String to Field," and Function 34, "Copy Field to String," allow an application to access extended attribute bytes (EABs) in a 3270 or 5250 presentation space. Information in this Appendix explains format and interpretation of EABs.

3270 Character Attributes

When a subject function is executed with session parameter EAB in effect, EAB data are passed to or from a 3270 presentation space in the following format:

Bit	Meaning
0–1	Character highlighting 00 = Normal 01 = Blink 10 = Reverse video 11 = Underline
2-4	Character color 000 = Default 001 = Blue 010 = Red 011 = Pink 100 = Green 101 = Turquoise 110 = Yellow 111 = White
5-7	Reserved

5250 Character Attributes

When a subject function is executed with session parameter EAB in effect, EAB data are passed to or from a 5250 presentation space in the following format:

Bit	Meaning
0	0 = normal image, 1 = reverse image
1	0 = no underline, 1 = underline
2	0 = no blink, 1 = blink
3	0 = no column separator, 1 = column separator
4-7	Reserved

Appendix E: Attachmate HLLAPI messages

Attachmate HLLAPI provides a Windows message option with Function 23, "Start Host Notification"; Function 50, "Start Keystroke Intercept"; and, when running in asynchronous mode, Function 90, "Send File"; and Function 91, "Receive File." Attachmate HLLAPI notifies the client that an update of the appropriate kind has occurred; that a keystroke has been intercepted; or that a file transfer has been completed by posting a Windows message to the client.

EXTRA! broadcasts a system-close message when it is about to shut down in response to a user's request to terminate Windows or terminate EXTRA!. If there are open host sessions at the time the user initiates the termination (terminal emulator or file transfer sessions, or HLLAPI connections, monitors, or keystroke interceptors), EXTRA! warns the user that the termination will release resources and asks for confirmation before terminating.

The messages themselves are registered with Windows. To be able to process the messages, the HLLAPI application must call the Windows routine RegisterWindowsMessage (<string>) at some point before making one of the above HLLAPI calls; for example, while processing the WM_CREATE message for the client window, or as part of the WinMain code.

The value returned from RegisterWindowMessage (<string>) can then be tested for in the client's WinProc. An example follows the discussion of messages.

XM_FILETRANSFERCOMPLETION

This message indicates that an asynchronous file transfer has finished.

Parameter	Type	Description
wParam (LOBYTE)	cPSID	This contains the session short name.
wParam (HIBYTE)		Reserved.
lParam (LOWORD)		This is the transfer sequence ID.
lParam (HIWORD)		This is the transfer completion code; it can be either FT_SUCCESS (HIWORD = 0) or a numeric value greater than 0 indicating a transfer failure.

XM_KEYSTROKEINTERCEPTED

This message notifies the application that a keystroke has been intercepted.

Parameter	Type	Description
wParam (LOBYTE)	cPSID	This contains the session short name.
wParam (HIBYTE)		Reserved.
lParam (LOWORD)		This is the key filter type. It will be one of the following HLLAPI identifiers: HLL_INTERCEPTAIDKEYS HLL_INTERCEPTALLKEYS
lParam (HIWORD)		Reserved.

XM_KILLFILETRANSFER

This message allows an application to abort an asychronous file transfer. It is sent by the HLLAPI client to the file transfer application.

Parameter	Type	Description
hWnd		This is the file transfer application handle; it is returned as the HIWORD from HLL_SENDFILE or HLL_RECEIVEFILE
wParam		This is the file transfer sequence ID; it is returned as a parameter from HLL_SENDFILE or HLL_RECEIVEFILE.
lParam		Reserved.

XM_PREVENTSYSTEMCLOSE

This message is returned to EXTRA! when EXTRA! asks if a client application can be shut down. By returning this message, the client application prevents EXTRA! from shutting down. EXTRA! asks by broadcasting a XM_QUERYSYSTEMCLOSE message to the client application.

Parameter	Type	Description
hWnd		This parameter uses wParam from the XM_QUERYSYSTEMCLOSE message.
wParam		Reserved.
lParam		Reserved.

XM_QUERYSYSTEMCLOSE

This message is sent to a client application by EXTRA! to ask if the application can be shut down. The applications can prevent EXTRA! from shutting down by sending back a XM_PREVENTSYSTEMCLOSE message.

Parameter	Type	Description
wParam	cPSID	This is the handle to be used in XM_PREVENTSYSTEMCLOSE.
lParam		Reserved.

XM_SESSIONUPDATE

This message notifies the client application window that an update has occurred in a session where the client application window called Function 23, "Start Host Notification."

Parameter	Type	Description
wParam (LOBYTE)	cPSID	This is the session short name.
lParam (LOWORD)		This is the type of update. Its value is a combination of the following identifiers from HLLAPI.H: HLL_NOTIFYPSUPDATE HLL_NOTIFYOIAUPDATE HLL_NOTIFYCURSORUPDATE HLL_NOTIFYBEEP HLL_NOTIFYBASECOLORCHANGE HLL_NOTIFYMODELCHANGE HLL_NOTIFYPOWERCHANGE
lParam (HIWORD)		This contains the session status prior to the current update. It's value is a combination of the following identifiers from HLLAPI.H: HLL_NOTIFYPSUPDATE

HLL_NOTIFYOIAUPDATE
HLL_NOTIFYCURSORUPDATE
HLL_NOTIFYBEEP
HLL_NOTIFYBASECOLORCHANGE
HLL_NOTIFYMODELCHANGE
HLL_NOTIFYPOWERCHANGE

XM_SYSTEMCLOSE

This message calls for an unconditional EXTRA! shutdown. All EXTRA! resources, including HLLAPI resources, will be released. HLLAPI clients, if they remain in operation after the termination of EXTRA!, no longer have access to HLLAPI resources. It is appropriate to call HLL_ResetHLLWin () at this point.

Parameter	Type	Description
wParam	cPSID	Reserved.
lParam		Reserved.

Example

To obtain the registered Windows message, the HLLAPI application must call the Windows routine RegisterWindowMessage () as follows:

```
WORD wmsgXMSYSTEMCLOSE, wmsgKEYSTROKE;
{
/* ...WinMain routine... */

wmsgXMSYSTEMCLOSE = RegisterWindowMessage ("XM_SYSTEMCLOSE");
wmsgKEYSTROKE = RegisterWindowMessage ("XM_KEYSTROKEINTERCEPTED");

...
}

/*If the WinProc is built around a large case statement */
/* pivoting on the message value, the test for wmsgXXXX */
/* can be made either before the switch (wMsg) */
/* or as part of the default case.*/

WinProc (
HWNDhWnd, WORD wMsg, WORD wParam, DWORD lparam)
{
       if (wMsg == wmsgXMSYSTEMCLOSE)
       {
              HLL_ResetHLLWin (hWnd);
              DestroyWindow (hWnd);
       }
       else
              switch (wMsg)
              {
              case XXXX:
                     break;
              case YYYY:
                     break;
              default:
                     if (wMsg == wmsgKEYSTROKEINTERCEPTED)
                     {
                            processKeystroke (wParam, lParam);
                     }
                     else
                            return DefWindowProc (hWnd, wMsg, wParam,
                            lParam);
              }
} /* end WinProc */
```