Windows HLLAPI Specification

Version 1.1

Greg Millard	**Digital Communications Associates, Inc.**
Sean Grinslade	**Attachmate Corporation**
David Fuchs	**Wall Data Incorporated**
Preston Sights	**Synapse Communications**
Michael Lee	**NetSoft**
Gordon Mangione	**Microsoft Corporation**

Microsoft Corporation

The specification was developed by the companies listed below (collectively, “Developers”). Although it is publicly available and is not confidential, the specification is still protected by copyright laws. Additional copies of the specification can be obtained on the MSDR forum on CompuServe® Information Services in Library 2.

This document is for informational purposes only. THE DEVELOPERS DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, IN THIS DOCUMENT. THE DEVELOPERS MAKE NO WARRANTY OR REPRESENTATION WITH RESPECT TO THIS SPECIFICATION, ITS QUALITY, PERFORMANCE, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. THE DEVELOPERS SHALL HAVE NO LIABILITY FOR SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT OF OR RESULTING FROM THE USE OR MODIFICATION OF THIS SPECIFICATION.

Contents

Chapter 1 Introduction 1
Windows HLLAPI Overview 1
IBM EHLLAPI 1
Microsoft Windows Graphical Environment and Windows Specific Extensions 2

Chapter 2 Programming with Windows HLLAPI 3
WinHLLAPI Installation Checking 3
Byte Ordering 3
Deviation from IBM EHLLAPI 3
Window Handle Passed for Each Async Call 4
Pointers 4
Blocking Routines 4

Chapter 3 Windows HLLAPI Functions 5
Windows Calls 6
Prerequisite Calls 7
5250 Emulation Support 8
Change Presentation Space Window Name—Function 105 10
Connect Presentation Space — Function 1 12
Connect Window Services—Function 101 13
Convert Position / RowCol—Function 99 14
Copy Field to String—Function 34 15
Copy OIA—Function 13 17
OIA Group Indicator Meanings for 5250 Sessions 22
Copy Presentation Space—Function 5 25
Copy Presentation Space to String—Function 8 27
Copy String to Field—Function 33 29
Copy String to Presentation Space—Function 15 31
Disconnect Presentation Space—Function 2 33
Disconnect Window Services—Function 102 34
Find Field Length—Function 32 35
Find Field Position—Function 31 37
Get Key—Function 51 39
Pause—Function 18 41
Post Intercept Status—Function 52 42

Query Close Intercept—Function 42 43
Query Cursor Location—Function 7 44
Query Field Attribute—Function 14 45
Query Host Update—Function 24 46
Query Session Status—Function 22 47
Query Sessions—Function 10 49
Query System—Function 20 50
Query Window Coordinates—Function 103 51
Receive File—Function 91 52
Asynchronous Mode 53
Release—Function 12 54
Reserve—Function 11 55
Reset System—Function 21 56
Search Field—Function 30 57
Search Presentation Space—Function 6 59
Send File—Function 90 61
Send Key—Function 3 63
Set Cursor—Function 40 67
Set Session Parameters—Function 9 68
STRLEN/STREOT 70
EOT=c 70
SRCHALL/SRCHFROM 71
SRCHFRWD/SRCHBKWD 71
NOATTRB/ATTRB 71
FPAUSE/IPAUSE 72
NOQUIET/QUIET 72
TIMEOUT=0/TIMEOUT=c 72
ESC=c 73
AUTORESET/NORESET 73
TWAIT/LWAIT/NWAIT 74
TRON/TROFF 74
EAB/NOEAB 74
XLATE/NOXLATE 75
CONLOG/CONPHYS 75
OLDOIA/NEWOIA 75
NOCFGSIZE/CFGSIZE 75
DISPLAY/NODISPLAY 76
WRITE_SUPER/WRITE_WRITE/WRITE_READ/WRITE_NONE/SUPER_WRITE/READ_WRITE 76

NOKEY/KEY$nnnnnnnn 77
Start Close Intercept—Function 41 78
Start Host Notification—Function 23 80
Start Keystroke Intercept—Function 50 82
Stop Close Intercept—Function 43 84
Stop Host Notification—Function 25 85
Stop Keystroke Intercept—Function 53 86
Wait—Function 4 87
Window Status—Function 104 89

Chapter 4 Extensions for the Windows Environment 93
WinHLLAPIAsync( ) 94
Windows HLLAPI Supplier Notes 94
WinHLLAPICleanup( ) 95
Windows HLLAPI Supplier Notes 95
WinHLLAPIIsBlocking( ) 96
Windows HLLAPI Supplier Notes 96
WinHLLAPICancelAsyncRequest( ) 97
WinHLLAPICancelBlockingCall( ) 98
WinHLLAPIStartup( ) 99
Windows HLLAPI Supplier Notes 101
WinHLLAPISetBlockingHook( ) 102
Windows HLLAPI Supplier Notes 103
WinHLLAPIUnhookBlockingHook( ) 104

Appendix A WHLLAPI.H - Definitions / Declarations for the Windows HLLAPI Specification 105

Appendix B Attributes 111
Character Attributes 112
Character Color Attributes 113
Field Attributes 114

Appendix C Extended Windows HLLAPI Functions 117
Allocate Communications Buffer—Function 123 118
Connect Structured Fields—Function 120 120
Disconnect Structured Fields—Function 121 123
Free Communications Buffer—Function 124 125
Get Request Completion—Function 125 126
Lock Presentation Space API—Function 60 130

Lock Window Services API—Function 61 132
Query Communication Buffer Size—Function 122 133
Read Structured Fields—Function 126 135
Storage Manager—Function 17 140
Get Storage 141
Free Storage 141
Free All Storage 142
Query Free Storage 142
Write Structured Fields—Function 127 143

Appendix D Query Reply Data Structures for Windows HLLAPI 149
The DDM Query Reply 150
DDM Application Name Self-Defining Parameter 150
PCLK Protocol Controls Self-Defining Parameter 151
Base DDM Query Reply Formats 151
The IBM Auxiliary Device Query Reply 153
Direct Access Self-Defining Parameter 155
PCLK Protocol Controls Self-Defining Parameter 155
The OEM Auxiliary Device Query Reply 156
Direct Access Self-Defining Parameter 156
PCLK Protocol Controls Self-Defining Parameter 157
The Cooperative Processing Requester Query Reply 158
The Product Defined Query Reply 159
Direct Access Self-Defining Parameter 160
The Document Interchange Architecture Query Reply 161
Direct Access Self-Defining Parameter 162

CHAPTER 1

Introduction

Windows HLLAPI Overview

Windows™ HLLAPI defines a standard and consistent IBM® EHLLAPI-style API for the 16- and 32-bit versions of the Microsoft® Windows graphical environment. It encompasses both familiar IBM EHLLAPI-style routines and a set of Windows-specific extensions designed to allow the programmer to take advantage of the message-driven nature of the Windows graphical environment.

This API has been designed to provide a standard to which application developers can program and network software vendors can conform. These API details constitute documentation for application software developers and a specification for network software vendors.

Network software that conforms to this Windows HLLAPI specification will be considered "Windows HLLAPI Compliant." To be Windows HLLAPI Compliant, a vendor must implement 100% of this Windows HLLAPI specification (functions listed in Appendix C - Extended Windows HLLAPI Functions are not required for compliancy). Suppliers of such interfaces shall be referred to as "Windows HLLAPI Suppliers."

Applications that are capable of exploiting any Windows HLLAPI implementation will be considered as having a "Windows HLLAPI Interface" and will be referred to as "Windows HLLAPI Applications."

IBM EHLLAPI

Windows HLLAPI has been built on the *de facto* IBM EHLLAPI programming standard. Windows HLLAPI is intended to provide maximum programming familiarity and to allow the simplified porting of existing EHLLAPI-based source code. The Windows HLLAPI is consistent with release 1.0 of IBM Extended Services for OS/2® EHLLAPI Programming Reference.

Microsoft Windows Graphical Environment and Windows Specific Extensions

This API has been designed for ALL implementations and versions of the Windows environment from and including version 3.0. It thus provides for Windows HLLAPI implementations and Windows HLLAPI applications in both 16- and 32-bit operating environments.

Windows HLLAPI makes provisions for multithreaded Windows-based processes, where a process contains one or more threads of execution. In the Win16 non-multithreaded world, a task corresponds to a process with a single thread. All references to threads in this document refer to actual "threads" in multithreaded Windows environments. In non-multithreaded environments (such as version 3.0), use of the term thread refers to a Windows process.

The extensions to the Windows environment included in Windows HLLAPI are provided for maximum programming compatibility among Windows version 3.*x* and Windows NT™ and optimum application performance in both environments.

CHAPTER 2

Programming with Windows HLLAPI

WinHLLAPI Installation Checking

To detect the presence of any Windows HLLAPI implementations on a system, an application that has been linked with the Windows HLLAPI Import Library can attempt to call the **WinHLLAPIStartup()** routine. Alternately, an application can examine the **$PATH** environment variable to search for instances of Windows HLLAPI API implementations (WHLLAPI.DLL). For each instance, it can issue a **LoadLibrary()** call and use the **WinHLLAPIStartup()** routine to discover implementation-specific data.

This version of the Windows HLLAPI API specification does not attempt to address explicitly the issue of multiple stacks/multiple concurrent Windows HLLAPI implementations. Nothing in the specification should be interpreted as restricting multiple WinHLLAPI DLLs from being present and from being used concurrently by one or more Windows HLLAPI application.

Byte Ordering

The Intel® byte ordering is like that of the DEC® VAX® and so differs from the Internet and 68000-type processor byte ordering. Take care in your programming to ensure the correct orientation.

Deviation from IBM EHLLAPI

There are a few limited instances where Windows HLLAPI diverts from strict adherence to the IBM EHLLAPI conventions. This deviation is due to the nature of the Windows graphical environment and the way it differs from other HLLAPI platforms.

Error constants are consistent with IBM EHLLAPI to maintain backward compatibility with existing software.

Window Handle Passed for Each Async Call

A Window handle has been added as the first parameter passed to the **WinHLLAPIAsync** entry point. This allows the **WinHLLAPIAsync** implementation to distinguish between HLLAPI applications.

Pointers

All pointers used by applications with WHLLAPI should be FAR.

Blocking Routines

Although blocking functions are supported with Windows HLLAPI, you should not use them. Instead you should use the **WinHLLAPIAsync** function in conjunction with a **WinHLLAPIAsync** Windows message.

CHAPTER 3

Windows HLLAPI Functions

Windows HLLAPI functions are requested using the appropriate parameter within the **WinHLLAPI()** call, and by specifying the function constant (or number equivalent) and the call parameters specific to that function. This chapter details the supported Windows HLLAPI functions, describing each function and their corresponding parameters and return codes. The supported functions are:

1 Connect Presentation Space
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
9 Set Session Parameters
10 Query Sessions
11 Reserve
12 Release
13 Copy OIA
14 Query Field Attribute
15 Copy String to Presentation Space
18 Pause
20 Query System
21 Reset System
22 Query Session Status
23 Start Host Notification
24 Query Host Update
25 Stop Host Notification
30 Search Field
31 Find Field Position
32 Find Field Length
33 Copy String to Field
34 Copy Field to String
40 Set Cursor Position
41 Start Close Intercept
42 Query Close Intercept
43 Stop Close Intercept
50 Start Keystroke Intercept
51 Get Key
52 Post Intercept Status
53 Stop Keystroke Intercept
90 Send File
91 Receive File
99 Convert Position / RowCol
101 Connect Window Services
102 Disconnect Window Services
103 Query Window Coordinates
104 Window Status
105 Change Switch List LT Name
106 Change PS Window Name

Windows Calls

The **WinHLLAPI( )** call requires you to specify four parameters in every call and has the following format:

extern VOID FAR PASCAL WinHLLAPI( )

LPWORD *lpwFunction*, /* Function name */

LPBYTE *lpbyString*, /* String pointer */

LPWORD *lpwLength*, /* String (data) length */

LPWORD *lpwReturnCode*); /* Return code */

Call Parameter Definitions

The parameters used in the WinHLLAPI function calls are:

Parameter	Description
lpwFunction	A pointer to the defined function name of the WinHLLAPI function call. It has a corresponding constant which you can find in WHLLAPI.H. To avoid additional complexity, the function prototype definitions for each function will list the constant instead of correctly listing a pointer to a word containing the function number.
lpbyString	A pointer to the *Data String* used by most Windows HLLAPI functions. Not all functions require a *Data String*. Some functions only use the *Data String* on the call (to pass data to **WinHLLAPI**), some only on the return (passing data back to your Windows HLLAPI application), and some on both the call and return.
lpwLength	This is a pointer to the *Data Length*, or it is the length of the *Data String*, depending on the particular Windows HLLAPI function call. Not all functions require a *Data Length*. Some functions only use the *Data Length* on the call (to pass a value to **WinHLLAPI**), some only on the return (passing a value back to your Windows HLLAPI application).
lpwReturnCode	A pointer to the return code. It indicates the status of the function request and is passed on the return from the function. Some functions, however, use this parameter on the call to indicate a position in the Host session presentation space. In this situation, the same parameter is referred to as "*PS Position*" on the call and "*Return Code*" on the return.

Prerequisite Calls

Most Windows HLLAPI functions require a prerequisite call—another function that must be called and successfully completed before the desired call can be issued. The following table lists the Windows HLLAPI functions and their prerequisite calls. “None” indicates that the function has no prerequisite call.

Function (Function Number)	Prerequisite Call (Function Number)
Connect Presentation Space (1)	None
Disconnect Presentation Space (2)	Connect Presentation Space (1)
Send Key (3)	Connect Presentation Space (1)
Wait (4)	Connect Presentation Space (1)
Copy Presentation Space (5)	Connect Presentation Space (1)
Search Presentation Space (6)	Connect Presentation Space (1)
Query Cursor Location (7)	Connect Presentation Space (1)
Copy Presentation Space to String (8)	Connect Presentation Space (1)
Set Session Parameters (9)	None
Query Sessions (10)	None
Reserve (11)	Connect Presentation Space (1)
Release (12)	Connect Presentation Space (1)
Copy OIA (13)	Connect Presentation Space (1)
Query Field Attribute (14)	Connect Presentation Space (1)
Copy String to Presentation Space (15)	Connect Presentation Space (1)
Pause (18)	None
Query System (20)	None
Reset System (21)	None
Query Session Status (22)	None
Start Host Notification (23)	None
Query Host Update (24)	Start Host Notification (23)
Stop Host Notification (25)	Start Host Notification (23)
Search Field (30)	Connect Presentation Space (1)
Find Field Position (31)	Connect Presentation Space (1)
Find Field Length (32)	Connect Presentation Space (1)
Copy String to Field (33)	Connect Presentation Space (1)
Copy Field to String (34)	Connect Presentation Space (1)
Set Cursor (40)	Connect Presentation Space (1)
Start Close Intercept (41)	None
Query Close Intercept (42)	Start Close Intercept (41)

Function (Function Number)	Prerequisite Call (Function Number)
Stop Close Intercept (43)	Start Close Intercept (41)
Start Keystroke Intercept (50)	None
Get Key (51)	Start Keystroke Intercept (50)
Post Intercept Status (52)	Start Keystroke Intercept (50)
Stop Keystroke Intercept (53)	Start Keystroke Intercept (50)
Send File (90)	None
Receive File (91)	None
Convert Position / RowCol (99)	None
Connect Window Services (101)	None
Disconnect Window Services (102)	Connect Window Services (101)
Query Window Coordinates (103)	Connect Window Services (101)
Window Status (104)	Connect Window Services (101)
Change PS Window Name (105)	Connect Window Services (101)

Note Some functions use the *Return Code* to pass a value to the call. This value is the Host session presentation space position. Although the parameter is listed as *lpwReturnCode* in the call syntax, the function listings in this Chapter refer to this parameter as *PS Position* on the call and *Return Code* on the return.

5250 Emulation Support

Most Windows HLLAPI functions are supported for both 3270 and 5250 emulators. The following functions are not supported for 5250 emulation:

Start Close Intercept (41)

Query Close Intercept (42)

Stop Close Intercept (43)

Send File (90)

Receive File (91)

Connect Window Services (101)

Disconnect Window Services (102)

Query Window Coordinates (103)

Window Status (104)

Change Presentation Space Window Name (105)

Connect Structured Fields (120)

Disconnect Structured Fields (121)
Query Communications Buffer Size (122)
Allocate Communications Buffer (123)
Free Communications Buffer (124)
Get Request Completion (125)
Read Structured Fields (126)
Write Structured Fields (127)

Change Presentation Space Window Name—Function 105

This function allows the application to specify a new name for the presentation space window or reset the presentation space window to the default name.

Prerequisite Functions

Connect Window Services (function 101).

Function Call

WinHLLAPI(CHANGEPSNAME,*lpbyString***,***lpwLength***,***lpwReturnnCode***)**

Call Parameters

Parameter	Description	
Data String	Window name - a 17-byte string with the following format:	
	Byte 1	Short name session ID, or space or null for the current session.
	Byte 2	A change request option value, select one of the following: **WHLL_CHANGEPSNAME_SET**. Change the window name. **WHLL_CHANGEPSNAME_RESET**. Reset the window name.
	Bytes 3-63	An ASCII string of 1 to 61 bytes including a terminator byte. The ASCII string must end with a NULL character. This string must contain a least one non-NULL character followed by a NULL character.
Data Length	3-63.	
PS Position	NA	

Return Codes

Code	Description
WHLLOK	The function was successful.
WHLLNOTCONNECTED	An invalid presentation space was specified, or was not connected for window services.
WHLLPARAMETERERROR	An invalid option was specified.
WHLLSYSERROR	The function failed due to a system error.
WHLLPSENDED	The session stopped.

Remarks

A string is ended at the first NULL character found. The NULL character overrides the specified string length. If the NULL character is not at the end of the specified length, the last byte at the specified length is replaced by a NULL character and the remainder of the Data String is lost. If the NULL character is found before the specified length, the string is truncated at that point and the remainder of the Data String is lost. This function is not supported for 5250 emulation.

Connect Presentation Space — Function 1

This function establishes a connection between a specified presentation space (session) on the Host and your Windows HLLAPI application.

Prerequisite Functions

None.

Function Call

WinHLLAPI(CONNECTPS,*lpbyString***,***lpwLength***,***lpwReturnnCode***)**

Call Parameters

Parameter	Description
Data String	One-character short name session ID of the Host session to connect with, either an upper- or lower-case letter.
Data Length	NA (defaults to 1).
PS Position	NA

Return Codes

Code	Description
WHLLOK	Connect request successful and the specified session is unlocked and ready for input.
WHLLNOTCONNECTED	Connect request failed, specified short name session ID is invalid.
WHLLPSBUSY	Connect request successful, but the specified session is busy.
WHLLINHIBITED	Connect request successful, but the specified session is locked (input inhibited).
WHLLSYSERROR	Connect request failed due to a system error.
WHLLUNAVAILABLE	Connect request failed, specified session is unavailable (already in use).

Remarks

When using the **WinHLLAPI( )** call, you can only make connected sessions available to other Windows HLLAPI applications by issuing a Disconnect Presentation Space call. Issuing a Reset System call causes your Windows HLLAPI application to disconnect from all Host sessions. Alternatively, your Windows HLLAPI application can share the Presentation Space by setting the appropriate read/write parameters in the Set Session Parameters call (Function 9).

Connect Window Services—Function 101

This function allows the application to manage the presentation space windows. Only one Windows HLLAPI application at a time can be connected to a presentation space for window services.

Prerequisite Functions

None.

Function Call

WinHLLAPI(CONNECTWINDOWSERVICES,*lpbyString***,** *lpwLength***,***lpwReturnnCode***)**

Call Parameters

Parameter	Description
Data String	One-character short name session ID of the presentation space.
Data Length	NA (defaults to 1).
PS Position	NA

Return Codes

Code	Description
WHLLOK	The function was successful.
WHLLNOTCONNECTED	An invalid presentation space was specified.
WHLLSYSERROR	The function failed due to a system error.
WHLLNOTSUPPORTED	The function was not supported by the emulation program.
WHLLUNAVAILABLE	The presentation space was being used by another function.

Remarks

A Windows HLLAPI application may connect to more than one presentation space concurrently for window services. More than one Windows HLLAPI application can share a presentation space, but the applications must synchronize session usage. This function is not supported for 5250 emulation.

Convert Position / RowCol—Function 99

This function converts a Host session presentation space position into row and column values for the PC display or converts PC display row and column values into a Host session presentation space position.

Prerequisite Functions

None.

Function Call

WinHLLAPI(CONVERT,*lpbyString***,***lpwLength***,***lpwReturnnCode***)**

Call Parameters

Parameter	Description
Data String	A 2-byte string. The first byte is the short name session ID of the Host session presentation space to convert. The second byte is "P" to convert a position to row and column, "R" to convert row and column to a position.
Data Length	NA when byte 2 of *Data String* is "P," row number when byte 2 of *Data String* is "R."
PS Position	Host session presentation space position value when byte 2 of *Data String* is "P," column number when byte 2 of *Data String* is "R."

Return Parameters

Parameter	Description
Data Length	When byte 2 of *Data String* on the call is "P," this value is the row number. When byte 2 of *Data String* on the call is "R," a zero indicates an invalid row number on the call.

Return Codes

Code	Description
WHLLOK	Invalid column number on the call.
>0	When byte 2 of *Data String* on the call is "P," this value is the column number. When byte 2 of *Data String* on the call is "R," this value is the Host session presentation space position.
WHLLINVALIDPSID	The function failed due to an invalid short name session ID or a system error.
WHLLINVALIDRC	Byte 2 in *Data String* on the call is invalid (neither "P" nor "R").

Remarks

If you need to determine the number of rows and columns that a Host session presentation space supports, use Query Session Status (function 22).

Copy Field to String—Function 34

This function copies the contents of a specified field in the Host session presentation space to a string. You can use Copy Field to String for either protected or unprotected fields.

Prerequisite Functions

Connect Presentation Space (function 1).

Function Call

WinHLLAPI(COPYFIELDTOSTRING,*lpbyString***,***lpwLength***,** *lpwReturnnCode***)**

Call Parameters

Parameter	Description
Data String	Buffer to hold the copied field. If session option EAB is set, must be defined at least twice the length of the field to be copied.
Data Length	Number of characters to copy.
PS Position	The position in the Host session presentation space of the field to be copied. This value may be any byte within the field, since the copy always starts at the beginning of the field.

Return Parameters

Parameter	Description
Data String	Data copied from the specified field. The first byte of *Data String* is the first byte in the specified field.

Return Codes

Code	Description
WHLLOK	The specified field was copied successfully.
WHLLNOTCONNECTED	Your Windows HLLAPI application is not currently connected to a Host session.
WHLLPARAMETERERROR	One or more call parameters are invalid.
WHLLTRUNCATED	The specified field was copied, but the data to be copied and *Data String* were not the same size. If *Data String* is smaller than the specified field, the remaining copy data is truncated.
WHLLPOSITIONERROR	*PS Position* is invalid.
WHLLSYSERROR	The function failed due to a system error.
WHLLNOFIELD	The Host session presentation space is unformatted.

Remarks

Position in the Host session presentation space is determined by starting in the upper left corner of the screen display (row 1, column 1). At the end of each screen display row, the next Host session presentation space position is column 1 of the following screen display row. This process continues until the end of the Host session presentation space (screen display) is reached.
This function is affected by the session options EAB/NOEAB, ATTRB/NOATTRB, DISPLAY/NODISPLAY, and XLATE/NOXLATE. See Set Session Parameters (function 9) for details. These session options have the following effect:

Session Option	Effect on this Function
NOATTRB	Unknown values are translated into spaces.
ATTRB	Unknown values are copied untranslated.
NOEAB	Extended attribute bytes are not copied to the string.
EAB, XLATE	Extended attribute bytes are copied and translated into CGA colors.
EAB, NOXLATE	Extended attribute bytes are returned.
DISPLAY	Data in a non-display field is copied to the target buffer.
NODISPAY	Data in a non-display field is copied as null characters to the target buffer.

See **Appendix B - Attributes** for descriptions of character, character color, and field attributes Information about the field to copy can be obtained with Find Field Position (function 31) and Find Field Length (function 32). The field is copied into *Data String* beginning with the first byte of the field and ends when one of the following occurs:

- The end of the field is reached.
- The end of *Data String* is reached.

Copy OIA—Function 13

This function returns the Host session Operator Information Area (OIA).

Prerequisite Functions

Connect Presentation Space (function 1).

Function Call

WinHLLAPI(COPYOIA,*lpbyString***,***lpwLength***,***lpwReturnnCode***)**

Call Parameters

Parameter	Description
Data String	String buffer to hold the OIA. Must be defined for at least 103 bytes.
Data Length	Length of *Data String* in characters.
PS Position	NA

Return Parameters

Parameter	Description
Data String	A 103-byte string containing a copy of the OIA.

Return Codes

Code	Description
WHLLOK	OIA copied successfully.
WHLLNOTCONNECTED	Your Windows HLLAPI application is not currently connected to a Host session.
WHLLPARAMETERERROR	OIA not copied; *Data Length* contains an invalid value.
WHLLPSBUSY	OIA copied successfully; Host session is busy.
WHLLINHIBITED	OIA copied successfully; Host session is locked (input inhibited).
WHLLSYSERROR	The function failed due to a system error.

Remarks

The 103-byte string returned in *Data String* contains three areas of information, as follows:

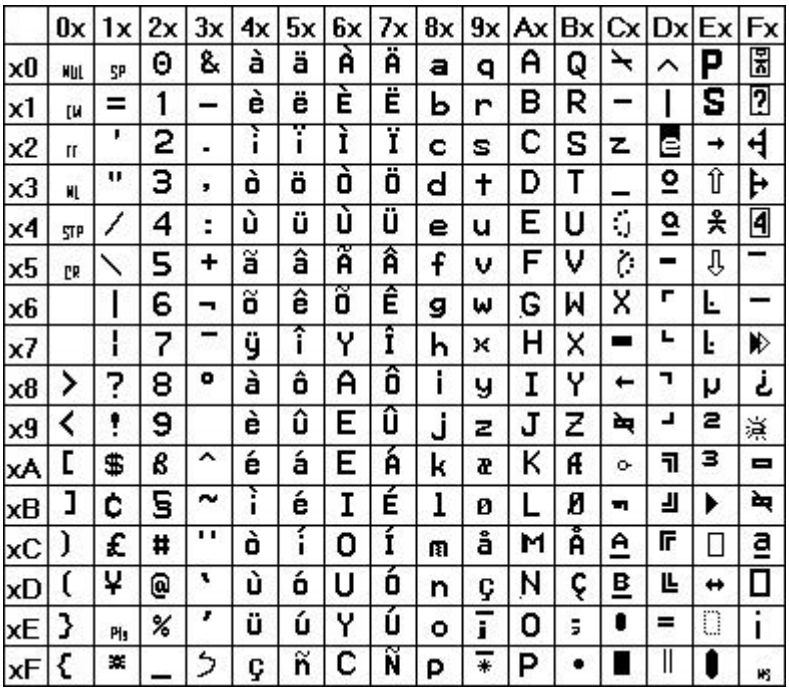
Byte 1	OIA format, “1” for 3270 or “9” for 5250.
Bytes 2-81	OIA image in binary format.
Bytes 82-103	OIA group indicator meanings. Unused positions are set to 00h.

Note The 5250 OIA image is always returned in ASCII. The 3270 OIA image will be returned in one of the following states depending on Set Session Parameter (9):

OLDOIA OIA image returned in 3270 PC format.

NEWOIA OIA image returned in ASCII format.

3270 Host Presentation Space Character Table



OIA Group Indicator Meanings for 3270 Sessions

Group 1: Online and screen ownership

1 byte (*Data String* position 82) applies to *Data String* position 2

Bits 0-1 Reserved

Bit 2 SSCP-LU session owns screen

Bit 3 LU-LU session owns screen

Bit 4 Online and not owned

Bit 5 Subsystem ready

Bits 6-7 Reserved

Group 2: Character selection

1 byte (*Data String* position 83) applies to *Data String* position 37

Bit 0 Reserved

Bit 1 APL

Bits 2-7 Reserved

Group 3: Shift state

1 byte (*Data String* position 84) applies to *Data String* position 43

Bit 0 Upper shift

Bit 1 Numeric

Bits 2-7 Reserved

Group 4: PSS group 1

1 byte (*Data String* position 85) not used (reserved)

Group 5: Highlight group 1

1 byte (*Data String* position 86) applies to *Data String* position 48

Bit 0 User-selectable

Bit 1 Field inherit

Bits 2-7 Reserved

Group 6: Color group 1

1 byte (*Data String* position 87) applies to *Data String* position 50

Bit 0 User-selectable

Bit 1 Field inherit

Bits 2-7 Reserved

Group 7: Insert

1 byte (*Data String* position 88) applies to *Data String* position 53

Bit 0 Insert mode

Bit 1-7 Reserved

Group 8: Input inhibited

5 bytes (*Data String* positions 89-93) apply to *Data String* position 10 (except where noted)

Byte 1 (*Data String* position 89)

Bit 0	Non-resetable machine check
Bit 1	Reserved
Bit 2	Machine check
Bit 3	Communications check
Bit 4	Program check
Bit 5	Reserved
Bit 6	Device not working
Bit 7	Reserved

Byte 2 (*Data String* position 90)

Bit 0	OIA time
Bit 1	Terminal wait
Bit 2	Reserved
Bit 3	Minus function
Bit 4	Too much entered
Bits 5-7	Reserved

Byte 3 (*Data String* position 91)

Bit 0	Reserved
Bit 1	User-unauthorized
Bit 2	User-unauthorized, minus function
Bit 3	Invalid dead key combination
Bit 4	Wrong place
Bits 5-7	Reserved

Byte 4 (*Data String* position 92)

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

Byte 5 (*Data String* position 93)

Bits 0-7	Reserved

Group 9: PSS group 2

1 byte (*Data String* position 94) not used (reserved)

Group 10: Highlight group 2

1 byte (*Data String* position 95)

Bit 0 Selected

Bit 1-7 Reserved

Group 11: Color group 2

1 byte (*Data String* position 96)

Bit 0 Selected

Bit 1-7 Reserved

Group 12: Communication error reminder applies to *Data String* position 23

1 byte (*Data String* position 97)

Bit 0 Communications error

Bit 1-7 Reserved

Group 13: Printer status applies to *Data String* position 62

1 byte (*Data String* position 98)

Bit 0 Reserved

Bit 1 Printer malfunction

Bit 2 Printer printing

Bit 3 Printer assignment

bit 4-7 Reserved

Group 14: Graphics

1 byte (*Data String* position 99) not used (reserved)

Group 15: Not used

1 byte (*Data String* position 100) not used (reserved)

Group 16: Autokey play/record status

1 byte (*Data String* position 101) not used (reserved)

Group 17: Autokey abort/pause status

1 byte (*Data String* position 102) not used (reserved)

Group 18: Enlarge state

1 byte (*Data String* position 103) not used (reserved)

OIA Group Indicator Meanings for 5250 Sessions

Group 1: Online and screen ownership

1 byte (*Data String* position 82) applies to *Data String* position 19

Bits 0-2 Reserved

Bit 3 System available

Bit 4 Reserved

Bit 5 Subsystem ready

Bits 6-7 Reserved

Group 2: Character selection

1 byte (*Data String* position 83) applies to *Data String* position 44

Bits 0-4 Reserved

Bit 5 Diacritic mode

Bit 6-7 Reserved

Group 3: Shift state

1 byte (*Data String* position 84) applies to *Data String* position 39

Bits 0 Reserved

Bit 1 Keyboard Shift

Bits 2-7 Reserved

Group 4: PSS group 1

1 byte (*Data String* position 85) not used (reserved)

Group 5: Highlight group 1

1 byte (*Data String* position 86) not used (reserved)

Group 6: Color group 1

1 byte (*Data String* position 87) not used (reserved)

Group 7: Insert

1 byte (*Data String* position 88) applies to *Data String* position 49

Bit 0 Insert mode

Bits 1-7 Reserved

Group 8: Input inhibited

5 bytes (*Data String* positions 89-93) apply to *Data String* position 58 (except where noted)

Byte 1 (*Data String* position 89)

Bit 0-7 Reserved

Byte 2 (*Data String* position 90)

Bit 0-7 Reserved

Byte 3 (*Data String* position 91)

Bits 0-4 Reserved

Bit 5 User input error (II)

Bits 3-7 Reserved

Byte 4 (*Data String* position 92)

Bits 0-1 Reserved

Bit 2 System wait

Bits 3-7 Reserved

Byte 5 (*Data String* position 93)

Bits 0-7 Reserved

Bit 5 User input error (II)

Bits 3-7 Reserved

Group 9: PSS group 2

1 byte (*Data String* position 94) not used (reserved)

Group 10: Highlight group 2

1 byte (*Data String* position 95) not used (reserved)

Group 11: Color group 2

1 byte (*Data String* position 96) not used (reserved)

Group 12: Communication error reminder applies to *Data String* position 29

1 byte (*Data String* position 97)

Bit 0-6 Reserved

Bit 7 Message Waiting (MW)

Group 13: Printer status
1 byte (*Data String* position 98) not used (reserved)
Group 14: Graphics
1 byte (*Data String* position 99) not used (reserved)
Group 15: Not used
1 byte (*Data String* position 100) not used (reserved)
Group 16: Autokey play/record status
1 byte (*Data String* position 101) not used (reserved)
Group 17: Autokey abort/pause status
1 byte (*Data String* position 102) not used (reserved)
Group 18: Enlarge state
1 byte (*Data String* position 103) not used (reserved)

Copy Presentation Space—Function 5

This function copies the contents of the current Host session's presentation space into a string buffer.

Prerequisite Functions

Connect Presentation Space (function 1).

Function Call

WinHLLAPI(COPYPS,*lpbyString***,***lpwLength***,***lpwReturnnCode***)**

Call Parameters

Parameter	Description
Data String	String that will contain the Host session presentation space. The string length must be defined as the maximum size of the presentation space. If the session option EAB has been set with Set Session Parameters (function 9), the string length must be defined to be at least twice the size of the presentation space.
Data Length	NA (the length of the presentation space is assumed).
PS Position	NA

Return Parameters

Parameter	Description
Data String	String containing the Host session presentation space.

Return Codes

Code	Description
WHLLOK	Host session presentation space successfully copied to *Data String*; the session is active and the keyboard is unlocked.
WHLLNOTCONNECTED	Your Windows HLLAPI application is not currently connected to a session.
WHLLPSBUSY	Host session presentation space successfully copied to *Data String*; the session is waiting for a Host response.
WHLLINHIBITED	Host session presentation space successfully copied to *Data String*; the keyboard is locked.
WHLLSYSERROR	The function failed due to a system error.

Remarks

This function copies the entire Host session presentation space to the supplied string. To copy only a portion of the presentation space, use Copy Presentation Space to String (function 8).

This function translates characters from EBCDIC to ASCII. The translation depends on the setting of the following session options:

Session Option	Effect on this Function
NOATTRB	Unknown values are translated into spaces.
ATTRB	Unknown values are copied untranslated.
NOEAB	Extended attribute bytes are not passed to the string.
EAB, XLATE	Extended attribute bytes are passed and translated into CGA colors.
EAB, NOXLATE	Extended attribute bytes are returned.
DISPLAY	Data in non-display fields is copied to the target buffer.
NODISPLAY	Data in non-display fields is copied a null characters to the target buffer.

See **Appendix B - Attributes** for descriptions of character, character color, and field attributes

Copy Presentation Space to String—Function 8

This function copies all or part of the Host session presentation space into a string buffer.

Prerequisite Functions

Connect Presentation Space (function 1).

Function Call

WinHLLAPI(COPYPSTOSTR,*lpbyString*,*lpwLength*,*lpwReturnnCode***)**

Call Parameters

Parameter	Description
Data String	String that will contain the specified portion of the Host session presentation space. **Note**: If the EAB option is set under Set Session Parameters (9) to include extended attribute bytes in the copy, the string must be defined as at least twice the size of the presentation space that is copied.
Data Length	Number of characters.
PS Position	Position in the Host session presentation space where the copying is to begin. Must be greater than zero and less than or equal to the maximum size of the Host session presentation space.

Return Parameters

Parameter	Description
Data String	String containing the specified portion of the Host session presentation space.

Return Codes

Code	Description
WHLLOK	The specified portion of the Host session presentation space successfully copied to *Data String*; the session is active and the keyboard is unlocked.
WHLLNOTCONNECTED	Your Windows HLLAPI application is not currently connected to a session.
WHLLPARAMETERERROR	*Data Length* of zero specified.
WHLLPSBUSY	The specified portion of the Host session presentation space successfully copied to *Data String*; the session is waiting for a Host response.
WHLLINHIBITED	The specified portion of the Host session presentation space successfully copied to *Data String*; the keyboard is locked.
WHLLPOSITIONERROR	*PS Position* value specified is invalid.
WHLLSYSERROR	The function failed due to a system error.

Remarks

PS Position in the Host session presentation space is determined by starting in the upper left corner of the screen display (row 1, column 1). At the end of each screen display row, the next Host session presentation space position is column 1 of the following screen display row. This process continues until the end of the Host session presentation space (screen display) is reached.
Character translation from EBCDIC to ASCII is performed by the Copy Presentation Space to String function. The translation depends on the setting of the following session options:

Session Option	Effect on this Function
NOATTRB	Unknown values are translated into spaces.
ATTRB	Unknown values are copied untranslated.
NOEAB	Extended attribute bytes are not copied to the string.
EAB, XLATE	Extended attribute bytes are copied and translated into CGA colors.
EAB, NOXLATE	Extended attribute bytes are returned.
DISPLAY	Data in non-display fields is copied to the target buffer.
NODISPLAY	Data in non-display fields is copied as null characters to the target buffer.

See **Appendix B - Attributes** for descriptions of character, character color, and field attributes

Copy String to Field—Function 33

This function copies a string into the specified field in the Host session presentation space.

Prerequisite Functions

Connect Presentation Space (function 1).

Function Call

WinHLLAPI(COPYSTRINGTOFIELD,*lpbyString***,***lpwLength***,** *lpwReturnnCode***)**

Call Parameters

Parameter	Description
Data String	String to copy to the specified field.
Data Length	Number of characters. NA if session option EOT is specified.
PS Position	The position in the Host session presentation space of the field to copy *Data String* to. This value can be any byte within the field, since the copy always starts at the beginning of the field.

Return Codes

Code	Description
WHLLOK	The string was copied successfully.
WHLLNOTCONNECTED	Your Windows HLLAPI application is not currently connected to a Host session.
WHLLPARAMETERERROR	*Data Length* invalid (set to zero).
WHLLINHIBITED	The specified field is protected, or attempted to copy invalid data (such as a field attribute).
WHLLTRUNCATED	The string was copied successfully, but one or more characters were truncated.
WHLLPOSITIONERROR	*PS Position* is invalid.
WHLLSYSERROR	The function failed due to a system error.
WHLLNOFIELD	The Host session presentation space is unformatted.

Remarks

Position in the Host session presentation space is determined by starting in the upper left corner of the screen display (row 1, column 1). At the end of each screen display row, the next Host session presentation space position is column 1 of the following screen display row. This process continues until the end of the Host session presentation space (screen display) is reached.

This function is affected by the session options STRLEN/STREOT, EOT=c and EAB/NOEAB. See Set Session Parameters (function 9) for details on these session options.

Data String is copied to the specified field, starting with the first position of the field, until one of the following occurs:

- The end of the field is encountered.
- If the EOT session option is set and an EOT is encountered.
- If the EOT session option is not set and the number of characters specified by *Data Length* have been copied.

Copy String to Presentation Space—Function 15

This function copies an ASCII string directly to a specified position in the Host session presentation space.

Prerequisite Functions

Connect Presentation Space (function 1)

Function Call

WinHLLAPI(COPYSTRTOPS,*lpbyString*,*lpwLength*,*lpwReturnnCode***)**

Call Parameters

Parameter	Description
Data String	String of ASCII data to copy to the Host session presentation space.
Data Length	Number of characters. NA if session option EOT is specified.
PS Position	Position in the Host session presentation space where *Data String* is to be copied.

Return Codes

Code	Description
WHLLOK	*Data String* successfully copied.
WHLLNOTCONNECTED	Your Windows HLLAPI application is not currently connected to a Host session.
WHLLPARAMETERERROR	*Data Length* has a value of zero (invalid).
WHLLINHIBITED	The Host session presentation space is protected or inhibited, or *Data String* contains illegal data (such as a field attribute byte).
WHLLTRUNCATED	*Data String* was truncated during the copy (partial copy).
WHLLPOSITIONERROR	*PS Position* is an invalid value.
WHLLSYSERROR	The function failed due to a system error.

Remarks

Position in the Host session presentation space is determined by starting in the upper left corner of the screen display (row 1, column 1). At the end of each screen display row, the next Host session presentation space position is column 1 of the following screen display row. This process continues until the end of the Host session presentation space (screen display) is reached.

This function is affected by the session options STRLEN/STREOT and EOT=c. If the session option STREOT has been specified, the copy string ends when an EOT is encountered in *Data String*. See Set Session Parameters (9) for details.

This function is similar to, but faster than, Send Key (3). However, keyboard mnemonics that can be sent with Send Key cannot be sent with this function.

Data String cannot be larger than the maximum size of the Host session presentation space.

5250 emulators supports a Presentation Space of 24 rows by 80 columns. When an error message from the host or when the operator presses the SysReq key, a 25th row is displayed. When the row 25 is displayed, it is a valid area for this function.

Disconnect Presentation Space—Function 2

This function disconnects a Host session from your Windows HLLAPI session.

Prerequisite Functions

Connect Presentation Space (function 1).

Function Call

WinHLLAPI(DISCONNECTPS,*lpbyString***,***lpwLength***,***lpwReturnnCode***)**

Call Parameters

Parameter	Description
Data String	NA
Data Length	NA
PS Position	NA

Return Codes

Code	Description
WHLLOK	The disconnect is successful.
WHLLNOTCONNECTED	Your Windows HLLAPI application is not currently connected to a Host session.
WHLLSYSERROR	The disconnect failed due to a system error.

Remarks

After calling this function, other functions that require a connected session are not valid and should not be called. The Windows HLLAPI application should disconnect from all connected sessions before exiting.

This function does not reset the session parameters to their defaults. In order to reset the default values, the Windows HLLAPI application must issue a Reset System (function 21).

Disconnect Window Services—Function 102

This function disconnects window services between a Windows HLLAPI application and a specified Windows HLLAPI session.

Prerequisite Functions

Connect Window Services (function 101).

Function Call

WinHLLAPI(DISCONNECTWINDOWSERVICES,*lpbyString***,** *lpwLength***,***lpwReturnnCode***)**

Call Parameters

Parameter	Description
Data String	One-character short name session ID of the presentation space.
Data Length	NA (defaults to 1).
PS Position	NA

Return Code

Code	Description
WHLLOK	The function was successful.
WHLLNOTCONNECTED	An invalid presentation space was specified, or was not connected for window services.
WHLLSYSERROR	The function failed due to a system error.

Remarks

After calling this function, other functions that require a connected session for window services are not valid and should not be called. The Windows HLLAPI application should disconnect from all sessions that have been connected for window services before exiting. This function is not supported for 5250 emulation.

Find Field Length—Function 32

This function determines the length of a specified field in the Host session presentation space. You can use Find Field Length for either protected or unprotected fields but only in a field-formatted host presentation space.

Prerequisite Functions

Connect Presentation Space (function 1).

Function Call

WinHLLAPI(FINDFIELDLENGTH,*lpbyString***,***lpwLength***,**
*lpwReturnnCode***)**

Call Parameters

Parameter	Description	
Data String	A 2-byte string that must be one of the following:	
	[space][space]	This (current) field.
	T[space]	This (current) field.
	P[space]	Previous field (protected or unprotected).
	N[space]	Next field (protected or unprotected).
	NP	Next Protected field.
	NU	Next Unprotected field.
	PP	Previous Protected field.
	PU	Previous Unprotected field.
Data Length	NA (length of 2 is implied).	
PS Position	The position in the Host session presentation space where the find starts.	

Return Parameters

Data Length	Description
0	If *Return Code* = 28, the field length is zero. If *Return Code* = 24, the Host session presentation space is unformatted.
>0	Length of the specified field. This value includes all characters from the beginning of the specified field up to the character preceding the next attribute byte.

Return Codes

Code	Description
WHLLOK	The specified field length was found.
WHLLNOTCONNECTED	Your Windows HLLAPI application is not currently connected to a Host session.
WHLLPARAMETERERROR	One or more of the call parameters are invalid.
WHLLPOSITIONERROR	*PS Position* is invalid.
WHLLSYSERROR	The function failed due to a system error.
WHLLNOFIELD	The specified field was not found, or the Host session presentation space is unformatted.
WHLLZEROLENFIELD	The specified field has a length of zero.

Remarks

Position in the Host session presentation space is determined by starting in the upper left corner of the screen display (row 1, column 1). At the end of each screen display row, the next Host session presentation space position is column 1 of the following screen display row. This process continues until the end of the Host session presentation space (screen display) is reached.
5250 emulators supports a Presentation Space of 24 rows by 80 columns. When an error message from the host or when the operator presses the SysReq key, a 25th row is displayed. When the row 25 is displayed, it is a valid area for this function.

Find Field Position—Function 31

This function determines the starting position of a field in the Host session presentation space. You can use Find Field Position for either protected or unprotected fields but only in a field-formatted host presentation space.

Prerequisite Functions

Connect Presentation Space (function 1).

Function Call

WinHLLAPI(FINDFIELDPOSITION,*lpbyString***,***lpwLength***,**
*lpwReturnnCode***)**

Call Parameters

Parameter	Description	
Data String	A 2-byte string that must be one of the following:	
	[space][space]	This (current) field.
	T[space]	This (current) field.
	P[space]	Previous field (protected or unprotected).
	N[space]	Next field (protected or unprotected).
	NP	Next Protected field.
	NU	Next Unprotected field.
	PP	Previous Protected field.
	PU	Previous Unprotected field.
Data Length	NA (length of 2 is implied).	
PS Position	The position in the Host session presentation space where the find starts.	

Return Parameters

Data Length	Description
0	If *Return Code* = 28, the field length is zero. If *Return Code* = 24, the Host session presentation space is unformatted.
>0	Starting position of the requested field.

Return Codes

Code	Description
WHLLOK	The specified field was found.
WHLLNOTCONNECTED	Your Windows HLLAPI application is not currently connected to a Host session.
WHLLPARAMETERERROR	One or more of the call parameters are invalid.
WHLLPOSITIONERROR	*PS Position* is invalid.
WHLLSYSERROR	The function failed due to a system error.
WHLLNOFIELD	The specified field was not found, or the Host session presentation space is unformatted.
WHLLZEROLENFIELD	The specified field has a length of zero.

Remarks

Position in the Host session presentation space is determined by starting in the upper left corner of the screen display (row 1, column 1). At the end of each screen display row, the next Host session presentation space position is column 1 of the following screen display row. This process continues until the end of the Host session presentation space (screen display) is reached.
5250 emulators supports a Presentation Space of 24 rows by 80 columns. When an error message from the host or when the operator presses the SysReq key, a 25th row is displayed. When the row 25 is displayed, it is a valid area for this function.

Get Key—Function 51

This function allows your Windows HLLAPI application to intercept keystrokes from Host sessions that have keystroke intercept enabled, and to process those keystrokes.

Prerequisite Functions

Start Keystroke Intercept (function 50).

Function Call

WinHLLAPI(GETKEY,*lpbyString***,***lpwLength***,***lpwReturnnCode***)**

Call Parameters

Parameter	Description	
Data String	An 8-byte string in the following format:	
	Byte 1	Short name session ID of the desired Host session, or space or null for the current Host session.
	Bytes 2-8	Reserved for return data.
Data Length	NA (length of 8 is implied).	
PS Position	NA.	

Return Parameters

Parameter	Description	
Data String	An 8-byte string in the following format:	
	Byte 1	Short name session ID of the desired Host session, or space or null for the current Host session.
	Byte 2	Keystroke code. “A” indicates an ASCII character; “M” indicates a 3270 function key code; “S” indicates a special key modifier (SHIFT, CTRL, or ALT) state.
	Bytes 3-8	Keystroke(s). Unused bytes are set to null (00h). See “Remarks” section for details.
Data Length	Number of characters in the returned mnemonic.	

Return Codes

Code	Description
WHLLOK	Keystroke(s) successfully returned.
WHLLNOTCONNECTED	The Host session presentation space is invalid.
WHLLINHIBITED	Start Keystroke Intercept (function 50) was called with the “D” option (intercept AID keys only). Non-AID keys are not returned.
WHLLNOTAVAILABLE	Start Keystroke Intercept (function 50) was not called prior to this function call.
WHLLSYSERROR	The function failed due to a system error.

Code	Description
WHLLUNDEFINEDKEY	The user entered an invalid key combination for this Host session presentation space.
WHLLNOKEYSTROKES	There are no keystrokes available in the keystroke queue.
WHLLKEYOVERFLOW	The keystroke queue has overflowed and keystroke(s) were lost.

Remarks

This function is affected by the session options ESC=c and NWAIT/LWAIT/TWAIT. See Set Session Parameters (function 9) for details. Of particular importance is the ESC=c session option: the escape character may be set to something other than the default of the at sign (@), which is used in the keystroke examples.

Keystrokes entered by the user are queued by WinHLLAPI. Use this function to read the keystrokes from the queue one at a time. You can then use Send Key (function 3) to pass the original keystrokes and/or any other keystrokes you want to send to the Host session presentation space.

The special key modifiers that can be returned indicate which key modifier is active:

@A	ALT key active.
@S	SHIFT key active.
@	CTRL key active.

The 3270 function key codes are defined under Send Key (function 3).

Returned Data String Examples

BAt	“B” is the short name session ID of the Host session. Returned keystroke is ASCII lowercase t (bytes 4-8 are null, 00h).
FM@2	“F” is the short name session ID of the Host session. Returned keystroke is the 3270 function key code for PF@ (bytes 5-8 are null, 00h).
KS@Aa	“K” is the short name session ID of the Host session. Returned keystroke with special key modifier is ALT+A (bytes 6-8 are null, 00h).
MS@rA	“M” is the short name session ID of the Host session. Returned keystroke with special key modifier is CTRL+SHIFT+A (bytes 6-8 are null, 00h). Note that because both the CTRL and SHIFT keys are active, only the CTRL key modifier is indicated but the SHIFT key is implicitly defined by the uppercase A.

Pause—Function 18

This function causes your application to wait for a specified amount of time.

Prerequisite Functions

None.

Function Call

WinHLLAPI(PAUSE,*lpbyString***,***lpwLength***,***lpwReturnnCode***)**

Call Parameters

Parameter	Description
Data String	NA
Data Length	Amount of time to pause in multiples of 0.5 seconds. For example, a value of 240 signifies 2 minutes (120 seconds).
PS Position	NA

Return Codes

Code	Description
WHLLOK	Pause completed (specified wait time has expired).
WHLLSYSERROR	The function failed due to a system error. Any time results are unpredictable.
WHLLPSCHANGED	The OIA or presentation space of the Host session has been updated. Use Query Host Update (function 24) for more information.

Remarks

You should use the Windows environment timer facility, WM_TIMER, instead of timing loops to wait for an event to occur. Note that by calling Start Host Notification (function 23) before this function, a Host event can terminate the Pause. When this happens, call Query Host Update (function 24) to determine which session had the update and the type of update.

This function is affected by the FPAUSE/IPAUSE session options. See Set Session Parameters (function 9) for details. If IPAUSE is set, the pending Host event satisfies the Pause call until Query Host Update (function 24) is completed.

Post Intercept Status—Function 52

This function notifies WinHLLAPI that a keystroke obtained with Get Key (function 51) has been accepted or rejected. If rejected, a beep is generated.

Prerequisite Functions

Start Keystroke Intercept (function 50).

Function Call

WinHLLAPI(POSTINTERCEPTSTATUS,*lpbyString***,***lpwLength***,** *lpwReturnnCode***)**

Call Parameters

Parameter	Description	
Data String	A 2-byte string in the following format:	
	Byte 1	Short name session ID of the desired Host session, or space or null for the current Host session.
	Byte 2	“A” to accept the keystroke; “R” to reject the keystroke.
Data Length	NA (length of 2 is implied).	
PS Position	NA.	

Return Codes

Code	Description
WHLLOK	The notification is successful.
WHLLNOTCONNECTED	The Host session presentation space is invalid.
WHLLPARAMETERERROR	One or more of the call parameters are invalid.
WHLLNOTAVAILABLE	Intercept (function 50) was not called prior to this function call.
WHLLSYSERROR	The function failed due to a system error.

Query Close Intercept—Function 42

This function allows the application to determine if the user selected to close the emulator program.

Prerequisite Functions

Start Close Intercept (function 41).

Function Call

WinHLLAPI(QUERYCLOSEINTERCEPT,*lpbyString***,***lpwLength***,** *lpwReturnnCode***)**

Call Parameters

Parameter	Description
Data String	One-character short name session ID of the presentation space.
Data Length	Must be specified.
PS Position	NA

Return Codes

Code	Description
WHLLOK	A close intercept event did not occur.
WHLLNOTCONNECTED	An invalid presentation space was specified.
WHLLPARAMETERERROR	An invalid option was specified.
WHLLNOTAVAILABLE	Start Close Intercept has not been called prior to this function for the specified presentation space.
WHLLSYSERROR	The function failed due to a system error.
WHLLPSENDED	The session stopped.
WHLLPSCHANGED	A close intercept event occurred.

Remarks

This function is not supported for 5250 emulation.

Query Cursor Location—Function 7

This function determines the location of the cursor in the Host session presentation space.

Prerequisite Functions

Connect Presentation Space (function 1).

Function Call

WinHLLAPI(QUERYCURSORLOC,*lpbyString***,***lpwLength***,**
*lpwReturnnCode***)**

Call Parameters

Parameter	Description
Data String	NA
Data Length	NA
PS Position	NA

Return Parameters

Parameter	Description
lpwLength	*Data Length*: the position of the cursor in the Host session presentation space.

Return Codes

Code	Description
WHLLOK	The cursor was successfully located.
WHLLNOTCONNECTED	Your Windows HLLAPI application is not currently connected.
WHLLSYSERROR	The function failed due to a system error

Remarks

5250 emulators supports a Presentation Space of 24 rows by 80 columns. When an error message from the host or when the operator presses the SysReq key, a 25th row is displayed. When the row 25 is displayed, it is a valid area for this function.

Query Field Attribute—Function 14

This function returns the attribute byte of the field containing the specified position in the Host session presentation space.

Prerequisite Functions

Connect Presentation Space (function 1).

Function Call

WinHLLAPI(QUERYFIELDATTRIBUTE,*lpbyString***,***lpwLength***,** *lpwReturnnCode***)**

Call Parameters

Parameter	Description
Data String	NA
Data Length	NA
PS Position	A position in the Host session presentation space that is within the field for which you want the attribute byte returned.

Return Parameters

Parameter	Description
Data Length	The attribute value if the screen is formatted. Zero if the screen is not formatted.

Return Codes

Code	Description
WHLLOK	Field attribute found successfully.
WHLLNOTCONNECTED	Your Windows HLLAPI application is not currently connected to a Host session.
WHLLPOSITIONERROR	Invalid value specified for *PS Position*.
WHLLSYSERROR	The function failed due to a system error.
WHLLNOFIELD	Field attribute not found due to unformatted Host session presentation space.

Remarks

Position in the Host session presentation space is determined by starting in the upper left corner of the screen display (row 1, column 1). At the end of each screen display row, the next Host session presentation space position is column 1 of the following screen display row. This process continues until the end of the Host session presentation space (screen display) is reached.

You must examine the attribute byte to determine all of the current field attributes. See **Appendix B - Attributes** for descriptions of field attributes

Query Host Update—Function 24

This function determines if the presentation space, Operator Information Area (OIA), or both, of the specified Host session have been updated since one of the following occurs:

- Start Host Notification (function 23) was called.
- The previous call of this function.

Prerequisite Functions

Start Host Notification (function 23).

Function Call

WinHLLAPI(QUERYHOSTUPDATE,*lpbyString***,***lpwLength***,** *lpwReturnnCode***)**

Call Parameters

Parameter	Description
Data String	Short name session ID of the desired Host session, or space or null for the current Host session.
Data Length	NA (length of 1 is implied).
PS Position	NA

Return Codes

Code	Description
WHLLOK	No updates.
WHLLNOTCONNECTED	The Host session specified is invalid.
WHLLNOTAVAILABLE	Start Host Notification (function 23) has not been called prior to this function for the specified Host session.
WHLLSYSERROR	The function failed due to a system error.
WHLLOIAUPDATE	One or more updates to the OIA of the specified Host session.
WHLLPSUPDATE	One or more updates to the presentation space of the specified Host session.
WHLLBOTHUPDATE	One or more updates to both the OIA and the presentation space of the specified Host session.

Query Session Status—Function 22

This function accesses the status of a specified session.

Prerequisite Functions

None.

Function Call

WinHLLAPI(QUERYSESSIONSTATUS,*lpbyString***,***lpwLength***,**
*lpwReturnnCode***)**

Call Parameters

Parameter	Description
Data String	18-byte string for returned session information; first byte is a short name session ID of the session to query, or space or null for the current session.
Data Length	18
PS Position	NA

Return Parameters

Parameter	Description
Data String	Session status—an 18-byte string with the following format:

Byte 1	Short name session ID.
Bytes 2-9	Long name session ID.
Byte 10	Session type: “D” for 3270 Host, “P” for personal computer, “F” for 5250 host, “G” for 5250 printer, and “E” for 3270 printer.
Byte 11	Sessions characteristics as a binary number explained below: 0 EAB 1 PSS 2-7 Reserved If bit 0 (EAB) = 0 the session has base attributes. If bit 0 (EAB) =1 the session has extended attributes If bit 1 (PSS) = 0 the session does not support programmed symbols. if bit 1 (PSS)=1 the session supports programmed symbols.

Bytes 12-13	Number of rows in the Host session presentation space. This is a binary number, not ASCII. If the session type is “E” or “G” (printers), the value is binary zero.

Parameter	Description	
	Bytes 14-15	Number of columns in the Host session presentation space. This is a binary number, not ASCII. If the session type is “E” or “G” (printers), the value is binary zero.
	Bytes 16-17	Host code page number, expressed as a binary number.
	Byte 18	Reserved.

Return Codes

Code	Description
WHLLOK	The requested session status is returned successfully.
WHLLNOTCONNECTED	The requested session is invalid.
WHLLPARAMETERERROR	*Data Length* is invalid.
WHLLSYSERROR	The function failed due to a system error.

Remarks

The rows and columns returned in *Data String* (positions 12-13 and 14-15) are the number of rows and columns that correspond to the Model type.

Query Sessions—Function 10

This function returns the number of Host screen sessions that are active, and a string containing information on each of the Host screen sessions. Host printer sessions are not supported.

Prerequisite Functions

None.

Function Call

WinHLLAPI(QUERYSESSIONS,*lpbyString***,***lpwLength***,***lpwReturnnCode***)**

Call Parameters

Parameter	Description
Data String	String buffer to hold the information string. Must be defined as 12*(number of active Host sessions) bytes.
Data Length	12*(number of active Host sessions).

Return Parameters

Parameter	Description	
Data String	12-byte fields (one for each active Host 3270 session) with session information, in the following format:	
	Byte 1	Short name session ID.
	Bytes 2-9	Long name session ID.
	Byte 10	Session Type (“H” for Host, “P” for personal computer).
	Bytes 11-12	Size of the presentation space expressed as a binary number (not ASCII).
Data Length	The number of active Host 3270 screen sessions.	

Return Codes

Code	Description
WHLLOK	Call successful.
WHLLPARAMETERERROR	*Data Length* is invalid.
WHLLSYSERROR	The function failed due to a system error.

Remarks

The return value of *Data Length* is set when the *Return Code* is 0 or 2. If you receive a *Return Code* of 2, use *Data Length* to recalculate the necessary value for the size of *Data String* (and the value for *Data Length* on the call).

Depending on the Session parameter specified the presentation size will vary:

CFGSIZE	The size of presentation space configured by the user.
NOCFGSIZE	The current size of the presentation space at the time the call is issued.

Query System—Function 20

This function determines the level and version of WHLLAPI under which your Windows HLLAPI application is running.

Prerequisite Functions

None.

Function Call

WinHLLAPI(QUERYSYSTEM,*lpbyString*,*lpwLength*,*lpwReturnnCode***)**

Call Parameters

Parameter	Description
Data String	Buffer for query data, must be defined for 35 bytes.
Data Length	NA (length of 35 is implied).
PS Position	NA

Return Parameters

Parameter	Description	
Data String	System status—a 35-byte string with the following format:	
	1	WinHLLAPI version number
	2-3	WinHLLAPI level number
	4-9	WinHLLAPI version date (mmddyy)
	10-12	Reserved
	13	Always “U”
	14	Always “E”
	15-16	WinHLLAPI product version number
	17-18	WinHLLAPI product level number
	19	Reserved
	20-23	Reserved
	24-27	Reserved
	28-29	Reserved
	30-31	Reserved
	32	Reserved
	33-35	Reserved

Return Codes

Code	Description
WHLLOK	The query completed successfully.
WHLLSYSERROR	The function failed due to a system error.

Query Window Coordinates—Function 103

This function requests the window coordinates for a presentation space.

Prerequisite Functions

Connect Window Services (function 101).

Function Call

WinHLLAPI(QUERYWINDOWCOORDINATES,*lpbyString***,** *lpwLength***,***lpwReturnnCode***)**

Call Parameters

Parameter	Description
Data String	17-byte string for returned session information; first byte is a short name session ID of the session to query, or space or null for the current session.
Data Length	NA (defaults to 17).
PS Position	NA

Return Parameters

Parameter	Description	
Data String	Window coordinates - a 17-byte string with the following format:	
	Byte 1	Short name session ID of the desired Host session, or space or null for the current Host session.
	Bytes 2-5	Specifies the x-coordinate of the upper-left corner of the window.
	Bytes 6-9	Specifies the y-coordinate of the lower-right corner of the window.
	Bytes 10-13	Specifies the x-coordinate of the lower-right corner of the window.
	Bytes 14-17	Specifies the y-coordinate of the upper-left corner of the window

Return Codes

Code	Description
WHLLOK	The function was successful.
WHLLNOTCONNECTED	An invalid presentation space was specified, or was not connected for window services.
WHLLPSENDED	The session stopped.

Remarks

The window coordinates are returned in pixels. This function is not supported for 5250 emulation.

Receive File—Function 91

This function transfers a file from the Host to the PC running the Windows HLLAPI application. The file transfer can be synchronous (dedicated) or asynchronous (call-and-return). See the "Remarks" section for information on asynchronous file transfer.

Prerequisite Functions

None.

Function Call

WinHLLAPI(hWnd,RECEIVEFILE,*lpbyString***,***lpwLength***,** *lpwReturnnCode***)**

WinHLLAPIAsync(*hWnd***,RECEIVEFILE,***lpbyString***,***lpwLength***,** *lpwReturnnCode***)**

Call Parameters

Parameter	Description
Data String	RECEIVE command parameters.
Data Length	Length of *Data String*. NA if session option EOT is specified.
PS Position	NA.

Return Codes

Code	Description
WHLLOK	File transfer started successfully (asynchronous mode only).
WHLLPARAMETERERROR	Parameter error or *Data Length* is zero or greater than 128.
WHLLFTXCOMPLETE	The file transfer completed (synchronous mode only).
WHLLFTXSEGMENTED	The file transfer completed with one or more segmented records (synchronous mode only).
WHLLSYSERROR	The function failed due to a system error.
WHLLFTXABORTED	The file transfer aborted, either due to the user entering CTRL+BREAK or (if a timeout was set by Set Session Parameters, function 9) because the timeout period expired.
WHLLINVALIDFUNCTIONNUM	Invalid function number.
WHLLFILENOTFOUND	PC file not found.
WHLLACCESSDENIED	Access denied to PC file.
WHLLMEMORY	Insufficient memory.
WHLLINVALIDENVIRONMENT	Invalid environment.
WHLLINVALIDFORMAT	Invalid format.

Remarks

This function is affected by the session options STRLEN/STREOT, EOT=c, QUIET/NOQUIET, and TIMEOUT=0/TIMEOUT=c. See Set Session Parameters (function 9) for details.
You cannot use this function on 5250 sessions, 5250 printer sessions, and 3270 printer sessions. Only one file transfer operation is supported at a time, regardless of the number of Host sessions accessed by your Windows HLLAPI application.
Data String should contain the RECEIVE command parameters that you would normally enter at the DOS prompt. For example, to receive the file SALES.RPT on your PC from the CMS file SLS REPRT A on the Host session with the short name session ID of "E:"

- *Data String* SALES.RPT E:SLS REPRT A (ASCII CRLF
- *Data Length* 35

Asynchronous Mode

When asynchronous mode is enabled by calling **WinHLLAPIAsync**, this function initiates the file transfer and immediately returns control to your Windows HLLAPI application. This frees your application to perform other tasks while the file transfer is occurring.
Since asynchronous mode returns control immediately, you must use Windows version 3.x message notification to determine the completion status of the file transfer. Use the **RegisterWindowsMessage( )** function to register the message "WinHLLAPIAsyncFileTransfer". The message notification is in the format:

(*wMsgID***,** *wParm***,** *lParm***)**

where

wMsgID	Is the message ID returned by **RegisterWindowsMessage**.
wParm	Is the status indicator: the high byte contains the short name session ID, the low byte contains the status. If the low byte is two, the file transfer is still in progress. If the low byte is three, the file transfer has completed.
lParm	Depends upon the low byte value of **wParm**. If the low byte of **wParm** is two (in progress), **lParm** is the number of bytes that have been transferred. If the low byte of **wParm** is three (completed), **lParm** is the two-digit Host TRANS code.

Release—Function 12

This function releases the currently Connected Host session presentation space locked with Reserve (function 11).

Prerequisite Functions

Connect Presentation Space (function 1).

Function Call

WinHLLAPI(RELEASE,*lpbyString***,***lpwLength***,***lpwReturnnCode***)**

Call Parameters

Parameter	Description
Data String	NA
Data Length	NA
PS Position	NA

Return Codes

Code	Description
WHLLOK	The Host session presentation space has been released (unlocked).
WHLLNOTCONNECTED	Your Windows HLLAPI application is not currently connected to a Host session.
WHLLSYSERROR	The function failed due to a system error.

Remarks

If you do not Release the Host session presentation space locked with Reserve (function 11), it remains locked until one of the following occurs:

- Your Windows HLLAPI application calls Disconnect Presentation Space (function 2).
- Your Windows HLLAPI application calls Reset System (function 21).

Reserve—Function 11

This function reserves the currently Connected Host session presentation space, locking out the user and preventing keyboard input.

Prerequisite Functions

Connect Presentation Space (function 1).

Function Call

WinHLLAPI(RESERVE,*lpbyString***,***lpwLength***,***lpwReturnnCode***)**

Call Parameters

Parameter	Description
Data String	NA
Data Length	NA
PS Position	NA

Return Codes

Code	Description
WHLLOK	The Host session presentation space has been reserved (locked).
WHLLNOTCONNECTED	Your Windows HLLAPI application is not currently connected to a Host session.
WHLLINHIBITED	The Host session is inhibited.
WHLLSYSERROR	The function failed due to a system error.

Remarks

Reserve locks out keyboard input. You can prevent the user from gaining access to the Host session with this function. Once the Host session presentation space is reserved, it remains locked until one of the following occurs:

- Your Windows HLLAPI application calls Release (function 12).
- Your Windows HLLAPI application calls Disconnect Presentation Space (function 2).
- Your Windows HLLAPI application calls Reset System (function 21).

Reset System—Function 21

This function reinitializes the system to its default (start) state:

- All session options are reset to their defaults.
- Event notification is stopped.
- Any reserved sessions are released.
- Connected sessions are disconnected.
- The current status of Host sessions is updated.

Prerequisite Functions

None.

Function Call

WinHLLAPI(RESETSYSTEM,*lpbyString***,***lpwLength***,***lpwReturnnCode***)**

Call Parameters

Parameter	Description
Data String	NA
Data Length	NA
PS Position	NA

Return Codes

Code	Description
WHLLOK	The system has been reset.
WHLLSYSERROR	The function failed due to a system error.

Remarks

This function is normally used at the beginning and end of a Windows HLLAPI application to reset the system to initial default conditions.

This function resets ALL connected sessions owned by the HLLAPI application. As a result, caution is advised when using this function.

Search Field—Function 30

This function searches a field in the Host session presentation space for the specified string. You can use Search Field for either protected or unprotected fields but only in a field-formatted host presentation space.

Prerequisite Functions

Connect Presentation Space (function 1).

Function Call

WinHLLAPI(SEARCHFIELD,*lpbyString***,***lpwLength***,***lpwReturnnCode***)**

Call Parameters

Parameter	Description
Data String	String to search for.
Data Length	Length of the search string. NA if session option EOT is specified.
PS Position	Position in the Host session presentation space of the field to search. If session options SRCHFROM and SRCHFRWD are set, indicates position to search from. If session options SRCHFROM and SRCHBKWD are set, indicates position to search to. If session option SRCHALL is set, can be the position of any byte in the field to search.

Return Parameters

Data Length	Description
0	The search string was not found.
>0	The search string was found. Value is the Host session presentation space position where the string begins.

Return Codes

Code	Description
WHLLOK	The search string was found.
WHLLNOTCONNECTED	Your Windows HLLAPI application is not currently connected to a Host session.
WHLLPARAMETERERROR	*Data String* was length zero or greater than the Host session presentation space size.
WHLLPOSITIONERROR	*PS Position* is invalid.
WHLLSYSERROR	The function failed due to a system error.
WHLLNOFIELD	The search string was not found, or the Host session presentation space is unformatted.

Remarks

Position in the Host session presentation space is determined by starting in the upper left corner of the screen display (row 1, column 1). At the end of each screen display row, the next Host session presentation space position is column 1 of the following screen display row. This process continues until the end of the Host session presentation space (screen display) is reached.
This function is affected by four session option parameters: STRLEN/STREOT, EOT=c, SRCHALL/SRCHFROM and SRCHFRWD/SRCHBKWD. See Set Session Parameters (9) for details on these session options. The first two parameters affect string length and termination, but the last two directly affect how the Host session presentation space is examined:

Session Option	Effect on this Function
SRCHALL	Searches the entire field, specified by *PS Position*, for the search string.
SRCHFROM, SRCHFRWD	The search begins at *PS Position* and moves to the end of the field. The search ends when the specified string is found or when the end of the field.
SRCHFROM, SRCHBKWD	The search begins at the end of the field and moves to the beginning of the field. The search ends when the specified string is found or when *PS Position* is reached.

Search Presentation Space—Function 6

This function allows you to search the Host session presentation space for a specified string.

Prerequisite Functions

Connect Presentation Space (function 1).

Function Call

WinHLLAPI(SEARCHPS,*lpbyString***,***lpwLength***,***lpwReturnnCode***)**

Call Parameters

Parameter	Description
Data String	String to search for.
Data Length	Length of the search string. NA if session option EOT is specified.
PS Position	Position in the Host session presentation space. If session options SRCHFROM and SRCHFRWD are set, indicates position to search from. If session options SRCHFROM and SRCHBKWD are set, indicates position to search to. NA if session option SRCHALL is set.

Return Parameters

Parameter	Description
Data Length	If equal to zero, indicates the string was not found. If greater than zero, indicates the Host session presentation space position where the string was found.

Return Codes

Code	Description
WHLLOK	The completed successfully. You must check the *Data Length* parameter to determine if the string was found.
WHLLNOTCONNECTED	Your Windows HLLAPI application is not currently connected to a session.
WHLLPARAMETERERROR	Invalid parameters were specified.
WHLLPOSITIONERROR	*PS Position* value is invalid.
WHLLSYSERROR	The function failed due to a system error.
WHLLNOFIELD	The string was not found.

Remarks

"Position" in the Host session presentation space is determined by starting in the upper left corner of the screen display (row 1, column 1). At the end of each screen display row, the next Host session presentation space position is column 1 of the following screen display row. This process continues until the end of the Host session presentation space (screen display) is reached.

This function is affected by four session option parameters: STRLEN/STREOT, EOT=c, SRCHALL/SRCHFROM and SRCHFRWD/SRCHBKWD. See Set Session parameters (function 9) for details on these session options. The first two parameters affect string length and termination, but the last two directly affect how the Host session presentation space is examined:

Session Option	Effect on this Function
SRCHALL, SRCHFRWD	Overrides *PS Position* parameter and searches from the beginning of the Host session presentation space for the specified string. If the string exists, the first instance of the string is returned.
SRCHALL, SRCHBKWD	Overrides *PS Position* parameter and searches from the end of the Host session presentation space for the specified string. If the string exists, the last instance of the string is returned.
SRCHFROM, SRCHFRWD	The search begins at *PS Position* and moves to the end of the Host session presentation space. The search ends when the specified string is found or when the end of the presentation space is reached.
SRCHFROM, SRCHBKWD	The search begins at the end of the Host session presentation space and moves to the beginning of the presentation space. The search ends when the specified string is found or when *PS Position* is reached.

This function can be used to determine when the Host session is available for input. If your Windows HLLAPI application is waiting for a specific message or prompt, issue this function until the message or prompt is found.

You can also use the SRCHFROM session option in combination with this function to find multiple occurrences of a string in the Host session presentation space.

Send File—Function 90

This function transfers a file from the PC running the Windows HLLAPI application to the Host. The file transfer can be synchronous (dedicated) or asynchronous (call-and-return). See the "Remarks" section for information on asynchronous file transfer.

Prerequisite Functions

None.

Function Call

WinHLLAPI(SENDFILE,*lpbyString***,***lpwLength***,***lpwReturnnCode***)**

WinHLLAPIAsync(*hWnd***,SENDFILE,***lpbyString***,***lpwLength***,**
*lpwReturnnCode***)**

Call Parameters

Parameter	Description
Data String	SEND command parameters.
Data Length	Length of *Data String*. NA if session option EOT is specified.
PS Position	NA.

Return Codes

Code	Description
WHLLOK	File transfer started successfully (asynchronous mode only).
WHLLPARAMETERERROR	Parameter error or *Data Length* is zero or greater than 128.
WHLLFTXCOMPLETE	The file transfer completed (synchronous mode only).
WHLLFTXSEGMENTED	Transfer completed with one or more segmented records (synchronous mode only).
WHLLSYSERROR	The function failed due to a system error.
WHLLTRANSABORTED	The file transfer aborted, either due to the user entering CTRL+BREAK or (if a timeout was set by Set Session Parameters, function 9) because the timeout period expired.
WHLLINVALIDFUNCTIONNUM	Invalid function number.
WHLLFILENOTFOUND	PC file not found.
WHLLACCESSDENIED	Access denied to PC file.
WHLLMEMORY	Insufficient memory.
WHLLINVALIDENVIRONMENT	Invalid environment.

Remarks

This function is affected by the session options STRLEN/STREOT, EOT=c, QUIET/NOQUIET, and TIMEOUT=0/TIMEOUT=c . See Set Session Parameters (function 9) for details.
You cannot use this function on 5250 sessions, 5250 printer sessions, or 3270 printer sessions. Only one file transfer operation is supported at a time, regardless of the number of Host sessions accessed by your Windows HLLAPI application.
Data String should contain the SEND command parameters that you would normally enter at the DOS prompt. For example, to send the file SALES.RPT from your PC to the CMS file SLS REPRT A on the Host session with the short name session ID of "E:"

- *Data String* SALES.RPT E:SLS REPRT A (ASCII CRLF
- *Data Length* 35

Asynchronous Mode

When asynchronous mode is enabled by calling **WinHLLAPIAsync**, the function initiates the file transfer and immediately returns control to your Windows HLLAPI application. This frees your application to perform other tasks while the file transfer is occurring.
Because asynchronous mode returns control immediately, you must use Windows version 3.x message notification to determine the completion status of the file transfer. Use the **RegisterWindowsMessage( )** function to register the message "WinHLLAPIAsyncFileTransfer". The message notification is in the format:

(*wMsgID***,** *wParm***,** *lParm***)**

where

wMsgID	Is the message ID returned by **RegisterWindowsMessage**.
wParm	Is the status indicator: the high byte contains the short name session ID, the low byte contains the status. If the low byte is zero, the file transfer is still in progress. If the low byte is one, the file transfer has completed.
lParm	Depends upon the low byte value of *wParm*. If the low byte of *wParm* is zero (in progress), *lParm* is the number of bytes that have been transferred. If the low byte of *wParm* is one (completed), *lParm* is the two-digit Host TRANS code.

Send Key—Function 3

This function sends one or more keystrokes (up to a maximum of 255) to the connected Host session. The keystrokes appear to the session as if they are entered by a user. The keystrokes can include host function keys and AID keys.

Prerequisite Functions

Connect Presentation Space (function 1).

Function Call

WinHLLAPI(SENDKEY,*lpbyString***,***lpwLength***,***lpwReturnnCode***)**

Call Parameters

Parameter	Description
Data String	String of keystrokes, maximum of 255 bytes (including host function key codes).
Data Length	Length of *Data String* in bytes. This parameter is overridden if in EOT mode.

Return Codes

Code	Description
WHLLOK	The keystrokes were sent successfully.
WHLLNOTCONNECTED	Your Windows HLLAPI application is not currently connected to a Host session.
WHLLPARAMETERERROR	The function call contains an invalid parameter.
WHLLPSBUSY	The session is busy; all of the keystrokes could not be sent.
WHLLINHIBITED	Input to the session is inhibited; keystrokes were rejected or invalid host function key codes were sent. All of the keystrokes could not be sent.
WHLLSYSERROR	The function failed due to a system error.

Remarks

You cannot send keystrokes to the Host session when the keyboard is locked or busy (input inhibited). You can check the keyboard status with Wait (function 4). It is also your responsibility to treat input-protected or numeric-only Host fields appropriately.

This function is affected by five session options specified by Set Session Parameters (function 9): AUTORESET/NORESET, STRLEN/STREOT, EOT=c, and ESC=c.

You can increase the performance of the Send Key function by setting the session option NORESET. If this session option is set to AUTORESET, a reset code is always added to the beginning of the keystroke string, resetting all states that can be reset (except input-inhibited states). The added reset code bytes are not deducted from the *Data String* length of 255.

By default, the length of the *Data String* parameter must be specified by the *Data Length* parameter. Optionally, you can implicitly define the *Data Length* parameter by using the EOT delimiter character, which is specified with Set Session Parameters (function 9).

Note Better character transfer performance is achieved with Copy String To Field (function 33) or Copy String To Presentation Space (function 15). However, only this function (Send Key) can send the host function keys.

This function can be used to send host function keys (including AID keys) to the Host by using special codes. These codes consist of an Escape character (default is "@," the "at" sign) and a mnemonic code that corresponds to the supported host functions. The desired host function key codes are included as

part of the *Data String* parameters. The Escape character can be changed with the session option ESC=c. See Set Session Parameters (function 9) for details.

When the *Data String* contains AID keys, the string includes characters up to, and including, the first AID key encountered. The segment string and segment length are set internally to the proper values as the segment is sent to the Host. Because some Host applications process AID keys differently, some keystrokes in a subsequent segment could be lost. It is therefore required that you do not create a *Data String* containing more than one AID key.

The characters that make up the host function key codes are part of the *Data String* and make up its total length. This means that you must be careful when using host function key codes to not exceed the maximum of 255 characters in the *Data String*. For example, if you need to send a string that contains the Enter key (code @E), then the two bytes for the Enter code must be included in the *Data Length* parameter.

The following table lists the host function keys and their corresponding codes. Please note that if a character is used in the code, the case of the character is important.

Meaning	Mnemonic	3270	5250
@	@@		X
Alt	@A	X	
Alternate Cursor	@$	X	X
Attention	@A@Q	X	X
Backspace	@<	X	X
Backtab (Left Tab)	@B	X	X
Clear	@C	X	X
Cmd Function Key	@A@Y		X
Cursor Down	@V		X
Cursor Left	@L		X
Cursor Right	@Z		X
Cursor Select	@A@J	X	
Cursor Up	@U	X	X
Delete	@D	X	X
Dup	@S@x	X	X
End	@q		X
Enter	@E	X	X
Erase EOF	@F	X	X
Erase Input	@A@F	X	X
Field Exit	@A@E		X
Field Mark	@S@y	X	X
Field -	@A@-		X
Field +	@A@+		X
Help	@H		X

Hexadecimal	@A@X		X
Home	@0 (zero)	X	X
Insert	@I	X7	X
Insert Toggle	@A@I		X
Host Print	@P		X
Left Tab (Back Tab)	@B	X	X
New Line	@N	X	X
Page Up	@u		X
Page Down	@v		X
Print (PC)	@A@t		X
Record Backspace	@A@<		X
Reset	@R	X	X
Right Tab (Tab)	@T	X	X
Shift	@S	X	
Sys Request	@A@H	X	X
Tab (Right Tab)	@T	X	X
Test	@A@C		X
PA1	@x	X	
PA2	@y	X	
PA3	@z	X	

Meaning	**Mnemonic**	**3270**	**5250**
PA4	@+	X	
PA5	@%	X	
PA6	@&	X	
PA7	@’	X	
PA8	@(	X	
PA9	@)	X	
PA10	@*	X	
PF1/F1	@1	X	X
PF2/F2	@2	X	X
PF3/F3	@3	X	X
PF4/F4	@4	X	X
PF5/F5	@5	X	X
PF6/F6	@6	X	X
PF7/F7	@7	X	X
PF8/F8	@8	X	X

PF9/F9	@9	X	X
PF10/F10	@a	X	X
PF11/F11	@b	X	X
PF12/F12	@c	X	X
PF13	@d	X	X
PF14	@e	X	X
PF15	@f	X	X
PF16	@g	X	X
PF17	@h	X	X
PF18	@i	X	X
PF19	@j	X	X
PF20	@k	X	X
PF21	@l	X	X
PF22	@m	X	X
PF23	@n	X	X
PF24	@o	X	X

Note If you want to use the “at” sign (@) in the *Data String*, you must use the two-byte code “@@”.

Set Cursor—Function 40

This function places the cursor at a specified position in the Host session presentation space.

Prerequisite Functions

Connect Presentation Space (function 1).

Function Call

WinHLLAPI(SETCURSOR,*lpbyString***,***lpwLength***,***lpwReturnnCode***)**

Call Parameters

Parameter	Description
Data String	NA.
Data Length	NA.
PS Position	Position in the Host session presentation space to locate the cursor.

Return Codes

Code	Description
WHLLOK	The cursor was successfully placed at the specified position.
WHLLNOTCONNECTED	Your Windows HLLAPI application is not currently connected to a Host session.
WHLLPSBUSY	The Host session is busy.
WHLLPOSITIONERROR	*PS Position* is invalid (less than 1 or greater than the maximum Host session presentation space size).
WHLLSYSERROR	The function failed due to a system error.

Remarks

5250 emulators supports a Presentation Space of 24 rows by 80 columns. When an error message from the host or when the operator presses the SysReq key, a 25th row is displayed. When the row 25 is displayed, it is a valid area for this function.

Set Session Parameters—Function 9

This function sets the options of the Host session. Session options that are not set with this function use their default values. Session options set with this function remain in effect until one of the following occurs:

- Another Set Session Parameters call sets a new value.
- Reset System (function 21) is called.

Prerequisite Functions

None.

Function Call

WinHLLAPI(SETSESSIONPARAMETERS,*lpbyString***,***lpwLength***,** *lpwReturnnCode*)

Call Parameters

Parameter	Description
Data String	String containing the desired session options to set. If more than one session option is set, use a comma or space to separate the session options. See “Remarks” section for an explanation of the session options.
Data Length	Explicit length of *Data String* (EOT cannot be used).
PS Position	NA

Return Parameters

Parameter	Description
Data Length	Remains the same as the call value if all session options in *Data String* are valid. If *Data String* contains any invalid session options, *Data Length* is set to the number of valid session options contained in *Data String*.

Return Codes

Code	Description
WHLLOK	All of the requested session options set as specified.
WHLLPARAMETERERROR	*Data String* contains one or more invalid session options.
WHLLSYSERROR	The function failed due to a system error.

Remarks

The following table lists the functions that are affected by session options, and the session options that affect them.

Function Name (function number)	Session Options
Connect Presentation Space (function 1)	CONLOG/CONPHYS, WRITE_SUPER/WRITE_WRITE /WRITE_READ/WRITE_NONE /SUPER_WRITE/READ_WRITE, NOKEY/KEY$nnnnnnn
Send Key (function 3)	STRLEN/STREOT, EOT=c, ESC=c, AUTORESET/NORESET, RETRY/NORETRY
Wait (function 4)	TWAIT/LWAIT/NWAIT
Copy Presentation Space (function 5)	NOATTRB/ATTRB, EAB/NOEAB, XLATE/NOXLATE, DISPLAY/NODISPLAY
Search Presentation Space (function 6)	STRLEN/STREOT, EOT=c, SRCHALL/SRCHFROM, SRCHFRWD/SRCHBKWD
Copy Presentation Space to String (function 8)	NOATTRB/ATTRB, EAB/NOEAB, XLATE/NOXLATE, DISPLAY/NODISPLAY
Query Sessions (function 10)	NOCFGSIZE/CFGSIZE
Copy OIA (function 13)	OLDOIA/NEWOIA
Copy String to Presentation Space (function 15)	STRLEN/STREOT, EOT=c, EAB/NOEAB
Pause (function 18)	FPAUSE/IPAUSE
Search Field (function 30)	STRLEN/STREOT, EOT=c, SRCHALL/SRCHFROM, SRCHFRWD/SRCHBKWD
Copy String to Field (function 33)	STRLEN/STREOT, EOT=c, EAB/NOEAB
Copy Field to String (function 34)	NOATTRB/ATTRB, EAB/NOEAB, XLATE/NOXLATE, DISPLAY/NODISPLAY

Function Name (function number)	Session Options
Get Key (function 51)	ESC=c, TWAIT/LWAIT/NWAIT
Send File (function 90)	STRLEN/STREOT, EOT=c, NOQUIET/QUIET, TIMEOUT=0/TIMEOUT=c
Receive File (function 91)	STRLEN/STREOT, EOT=c, NOQUIET/QUIET, TIMEOUT=0/TIMEOUT=c
Connect PM Window Service (function 101)	WRITE_SUPER/WRITE_WRITE /WRITE_READ/WRITE_NONE /SUPER_WRITE/READ_WRITE

The session options are described on the following pages, grouped by function. Each group of session options lists their general function, which Windows HLLAPI functions they affect, and the characteristics of each session option setting.

STRLEN/STREOT

Specify how the length of *Data String* is determined. Applies to Send Key (3), Search Presentation Space (6), Copy String to Presentation Space (15), Search Field (30), Copy String to Field (33), Send File (90) and Receive File (91).

Session Option	Description
STRLEN	*Data Length* explicitly defined. This is the default setting.
STREOT	*Data Length* not necessarily defined; *Data String* parameter on a function call ends in an EOT character.

EOT=c

When the STREOT session option is set, specify the delimiter character to mark the end of the *Data String* parameter on a function call. Applies to Send Key (3), Search Presentation Space (6), Copy String to Presentation Space (15), Search Field (30), Copy String to Field (33), Send File (90) and Receive File (91).

Session Option	Description
EOT=c	Set the EOT character to “c,” which must be a 1-byte literal character. There must not be a space on either side of the equal sign (“space” is not a valid EOT character). The default EOT character is binary zero.

SRCHALL/SRCHFROM

Determine how the Host session presentation space is to be searched. Applies to Search Presentation Space (6) and Search Field (30).

Session Option	Description
SRCHALL	If using Search Presentation Space (6), search the entire Host session presentation space. If using Search Field (30), search the entire field. SRCHALL is the default setting.
SRCHFROM	If session option SRCHFRWD is set, start search at specified *PS Position* and stop at the end (of the field or Host session presentation space). If session option SRCHBKWD is set, start search at the end and stop at specified *PS Position*.

SRCHFRWD/SRCHBKWD

When the SRCHFROM session option is set, determine the direction of the search. Applies to Search Presentation Space (6) and Search Field (30).

Session Option	Description
SRCHFRWD	When session option SRCHFROM is set, start search at specified *PS Position* and stop at the end (of the field or Host session presentation space). This is the default setting.
SRCHBKWD	When session option SRCHFROM is set, start search at the end (of the field or Host session presentation space) and stop at specified *PS Position*.

NOATTRB/ATTRB

Determine how to translate attributes to your Windows HLLAPI application. Applies to Copy Presentation Space (5), Copy Presentation Space to String (8) and Copy Field to String (34).

Session Option	Description
NOATTRB	Translate EBCDIC bytes that do not have ASCII equivalents to spaces (ASCII 20h). NOATTRB is the default setting.
ATTRB	EBCDIC bytes that do not have ASCII equivalents are not translated, but are passed as their original EBCDIC values.

FPAUSE/IPAUSE

Determine the type of pause to use. Applies to Pause (18).

Session Option	Description
FPAUSE	Full pause; pause for the length of time specified in Pause (function 18). FPAUSE is the default setting.
IPAUSE	Interruptible pause; once a Start Host Notification (function 23) call is made, any Host event ends the pause.

NOQUIET/QUIET

Determine whether the file transfer functions SEND FILE (90) and RECEIVE FILE (91) will generate messages displayed to the user. These options are not supported for 5250 emulation.

Session Option	Description
NOQUIET	SEND and RECEIVE messages are displayed.
QUIET	SEND and RECEIVE messages are not displayed.

TIMEOUT=0/TIMEOUT=c

Set the timeout interval to be used during file transfer operations. If a timeout occurs, the file transfer aborts. These options are not supported for 5250 emulation.

Session Option	Description
TIMEOUT=0	Set the timeout to 30 seconds. There must not be a space on either side of the equal sign; the value is a zero. TIMEOUT=0 is the default setting.

Session Option	Description
TIMEOUT=c	Set the timeout to a specific period. A CTRL+BREAK is issued automatically after the specified period. There must not be a space on either side of the equal sign; c is a one-byte character that can be only one of the following values: 1 30 seconds (0.5 minutes) 2 60 seconds (1.0 minutes) 3 90 seconds (1.5 minutes) 4 120 seconds (2.0 minutes) 5 150 seconds (2.5 minutes) 6 180 seconds (3.0 minutes) 7 210 seconds (3.5 minutes) 8 240 seconds (4.0 minutes) 9 270 seconds (4.5 minutes) J 300 seconds (5.0 minutes) K 330 seconds (5.5 minutes) L 360 seconds (6.0 minutes) M 390 seconds (6.5 minutes) N 420 seconds (7.0 minutes)

ESC=c

Specify the escape character to use for 3270 function key codes. Applies to Send Key (3) and Get Key (51).

Set the escape character to use for 3270 function key codes to c which is a one-byte literal character. There must not be a space on either side of the equal sign (“space” is not a valid escape character). The default escape character is the at sign (@).

AUTORESET/NORESET

Determine if Send Key (function 3) sends a reset prior to the keystroke string or not.

Session Option	Description
AUTORESET	A reset precedes the keystroke string specified with a Send Key (3), attempting to reset any states that can be reset (except input-inhibited). AUTORESET is the default setting.
NORESET	A reset does not precede the keystroke string specified with a Send Key (3).

TWAIT/LWAIT/NWAIT

Determine the characteristics of a wait period. Applies to Wait (4) and Get Key(51).

Session Option	Description
TWAIT	For Wait (4), wait up to 60 seconds before timing out on XCLOCK or XSYSTEM. For Get Key (function 51), wait until a keystroke is queued before returning. TWAIT is the default setting.
LWAIT	For Wait (4), wait until the XCLOCK or XSYSTEM clears. This setting is not recommended because your Windows HLLAPI does not regain control until the Host is available. For Get Key (51), wait until a keystroke is queued before returning.
NWAIT	No wait period applies. Wait (4) and Get Key (51) calls each check their respective status and return immediately.

TRON/TROFF

Determine whether to enable or disable Windows HLLAPI tracing. The information in the trace is intended to help debug a Windows HLLAPI program. Tracing is turned off when the Windows HLLAPI program ends or when TROFF is specified.

Session Option	Description
TROFF	Turn tracing off.
TRON	Turn tracing on. With tracing enabled, all executed Windows HLLAPI functions are traced.

EAB/NOEAB

Determine whether to include extended attributes (EABs) or not. Applies to Copy Presentation Space (5), Copy Presentation Space to String (8), Copy String to Presentation Space (15), Copy String to Field (33) and Copy Field to String (34).

Session Option	Description
EAB	Include extended attributes (EABs) with *Data String*. Since there is an EAB for every character that displays, you must define *Data String* to be twice the size of the Host session presentation space or field.
NOEAB	Do not include any extended attributes (EABs) with *Data String* (no EABs). This is the default setting.

XLATE/NOXLATE

Determine the translation of extended attributes (EABs). Applies to Copy Presentation Space (5), Copy Presentation Space to String (8) and Copy Field to String (34).

Session Option	Description
XLATE	Translate extended attributes (EABs) into CGA colors.
NOXLATE	Do not translate extended attributes (EABs).

CONLOG/CONPHYS

Specify which application will be the foreground application when connecting to a session. Applies to Connect Presentation Space (1).

Session Option	Description
CONLOG	After connection, your Windows HLLAPI application remains the foreground application. CONLOG is the default setting.
CONPHYS	After connection, the specified session becomes the foreground application, updating the session and accepting keyboard input. Your Windows HLLAPI application can still access the session (to monitor for a specific event, for example). To return foreground control to your Windows HLLAPI application, call Disconnect Presentation Space (2).

OLDOIA/NEWOIA

Specify the format for the data returned from Copy OIA (13).

Session Option	Description
OLDOIA	Data returned in 3270 PC format. For 5250 support the OIA is always returned in ASCII therefore OLDOIA is accepted but ignored.
NEWOIA	Data is returned in ASCII format.

NOCFGSIZE/CFGSIZE

Determine the presentation space size returned by Query Sessions (10).

Session Option	Description
NOCFGSIZE	Returns the current size of the connected presentation space.
CFGSIZE	Returns the configured size of the presentation space thereby ignoring any override of the presentation space by the host.

DISPLAY/NODISPLAY

Specify whether nondisplay fields will be copied using Copy Presentation Space (5), Copy Presentation Space to String (8), Copy OIA (13), Copy String to Presentation Space (15), Copy String to Field (33), and Copy Field to String (34).

Session Option	Description
DISPLAY	Nondisplay fields are copied to the target buffer in the same manner as display fields.
NODISPLAY	Nondisplay fields are copied as a string of nulls to the target buffer. This allows applications to display the copied buffer in the presentation window without displaying confidential information, such as passwords.

WRITE_SUPER/WRITE_WRITE/WRITE_READ /WRITE_NONE/SUPER_WRITE/READ_WRITE

Specify whether a Windows HLLAPI application can or will share the presentation space to which it is connected with another application using Connect Presentation Space (1) and Connect PM Window Services (101).

Session Option	Description
WRITE_SUPER	Set by a Windows HLLAPI application that requires write access and allows only supervisory applications to connect to its presentation space.
WRITE_WRITE	Set by a Windows HLLAPI application that requires write access and allows other applications that have predictable behavior to connect to its presentation space.
WRITE_READ	Set by a Windows HLLAPI application that requires write access and allows other applications to use read-only functions on its connected presentation space.
WRITE_NONE	Set by a Windows HLLAPI application that requires exclusive access to the connected presentation space. No other applications, not even supervisory, will have access to its presentation space.
SUPER_WRITE	Set by a Windows HLLAPI supervisory application allowing applications with write access to share the connected presentation space. The application setting this parameter will not cause errors for other application but provide only supervisory-type functions.
READ_WRITE	Set by a Windows HLLAPI application that requires read-only access and allows other applications that perform read-only functions to connect to its presentation space.

NOKEY/KEY$nnnnnnnn

Allow applications that have sharing requirements to limit access to a partner application (i.e. an application developed to work with it).

Session Option	Description
NOKEY	Allows the application to be compatible with existing applications that do not specify the KEY parameter.
KEY$nnnnnnn	Specify the keyword to restrict sharing of the presentation space. The keyword must be exactly 8-bytes long.

Start Close Intercept—Function 41

This function allows the application to intercept user requests to close the emulation program.

Prerequisite Functions

None.

Function Call

WinHLLAPI(STARTCLOSEINTERCEPT,*lpbyString***,***lpwLength***,** *lpwReturnnCode***)**

WinHLLAPIAsync(*hWnd***,STARTCLOSEINTERCEPT,***lpbyString***,** *lpwLength***,***lpwReturnnCode***)**

Call Parameters

Parameter	Description
Data String	A 5-byte string for returned semaphore address. The first byte is a short name session ID of the session to query, or space or null for the current session.
Data Length	Must be specified
PS Position	NA

Return Parameters

Parameter	Description
Data String	A 5-byte string with the following format: Byte 1 Short name session ID, or space or null for the current session. Bytes 2-5 Semaphore address.

Return Code

Code	Description
WHLLOK	The function was successful.
WHLLNOTCONNECTED	An invalid presentation space was specified.
WHLLPARAMETERERROR	An invalid option was specified.
WHLLSYSERROR	The function failed due to a system error.
WHLLNOTSUPPORTED	The function was not supported by the emulation program.
WHLLCANCEL	The asynchronous function was cancelled.

Remarks

Initially, the semaphore is set. After using this function, close requests from the user are discarded and the semaphore is cleared. Your application program can use the Query Close Intercept function to determine when a close request has occurred. This function is not supported for 5250 emulation.

Asynchronous Mode

When asynchronous mode is enabled by calling **WinHLLAPIAsync**, the function initiates close intercept and immediately returns control to your Windows HLLAPI application. This frees your application to perform other tasks while waiting for close requests.

Because asynchronous mode returns control immediately, you must use Windows version 3.x message notification to determine when close requests have occurred. Use the **RegisterWindowsMessage( )** function to register the message “WinHLLAPIAsync”. See **WinHLLAPIAsync** in Chapter 4 for details.

Start Host Notification—Function 23

This function enables notifying your Windows HLLAPI application of changes in the Host session presentation space or Operation Information Area (OIA).

Prerequisite Functions

None.

Function Call

WinHLLAPI(STARTHOSTNOTIFICATION,*lpbyString***,***lpwLength***,** *lpwReturnnCode***)**

WinHLLAPIAsync(*hWnd***,STARTHOSTNOTIFICATION,***lpbyString***,** *lpwLength***,***lpwReturnnCode***)**

Call Parameters

Parameter	Description	
Data String	A 7-byte string in the following format:	
	Byte 1	Short name session ID of the desired Host session, or space or null for the current Host session.
	Byte 2	Notification mode. “P” for presentation space update only, “O” for OIA update only, “B” for both presentation space and OIA updates. When calling WinHLLAPIAsync, this position can be “A”.
	Bytes 3-6	Not used; no error occurs if an old Windows HLLAPI application uses these positions.
	Byte 7	Reserved or replace with one of the following if using WinHLLAPIAsync and “A” in byte 2: “P” for presentation space update only, “O” for OIA update only, “B” for both presentation space and OIA updates
Data Length	Length of Host event buffer (256 recommended).	
PS Position	NA	

Return Parameters

Parameter	Description
Data String	Same as *Data String* on the call.

Return Codes

Code	Description
WHLLOK	Host notification enabled.
WHLLNOTCONNECTED	The specified Host session is invalid.
WHLLPARAMETERERROR	One or more parameters are invalid.
WHLLSYSERROR	The function failed due to a system error.
WHLLCANCEL	The asynchronous function was cancelled.

Remarks

Once enabled, Host notification is enabled until you call Stop Host Notification (function 25).
Once you call this function, you can use Pause (function 18) to notify your Windows HLLAPI application when the presentation space and/or OIA of a Host session have been updated. Use Query Host Update (function 24) to determine which parts of the Host session (presentation space, OIA, or both) have been updated.

Asynchronous Mode

When asynchronous mode is enabled by calling **WinHLLAPIAsync**, the function initiates host notification and immediately returns control to your Windows HLLAPI application. This frees your application to perform other tasks while waiting for host updates.
Because asynchronous mode returns control immediately, you must use Windows version 3.x message notification to determine when host updates have occurred. Use the **RegisterWindowsMessage( )** function to register the message “WinHLLAPIAsync”. See **WinHLLAPIAsync** in Chapter 4 for details.

Start Keystroke Intercept—Function 50

This function enables your Windows HLLAPI application to intercept keystrokes sent to a session by the user.

Prerequisite Functions

None.

Function Call

WinHLLAPI(STARTKSINTERCEPT,*lpbyString***,***lpwLength***,** *lpwReturnnCode***)**

WinHLLAPIAsync(*hWnd***,STARTKSINTERCEPT,***lpbyString***,***lpwLength***,** *lpwReturnnCode***)**

Call Parameters

Parameter	Description
Data String	A 6-byte string in the following format: 1 Short name session ID of the desired Host session, or space or null for the current Host session. 2 Keystroke intercept code. “D” causes only AID keystrokes to be intercepted; “L” causes all keystrokes to be intercepted. 3-6 Reserved.
Data Length	Variable (256 is recommended).
PS Position	NA.

Return Codes

Code	Description
WHLLOK	Keystroke intercept has been enabled.
WHLLNOTCONNECTED	The Host session presentation space is invalid.
WHLLPARAMETERERROR	One or more call parameters are invalid.
WHLLPSBUSY	The Host session is busy.
WHLLSYSERROR	The function failed due to a system error.
WHLLCANCEL	The asynchronous function was cancelled.

Remarks

Once this function is called, the intercepted keystrokes can be:

- Received with Get Key (function 51) and sent to the same session or another session with Send Key (function 3).
- Accepted and rejected with Post Intercept Status (function 52).
- Replaced by other keystrokes with Send Key (function 3).
- Used in a specific manner as appropriate for your Windows HLLAPI application.

If position 2 of *Data String* is “D,” only AID keystrokes are intercepted. All other keystrokes are passed on to the appropriate Host session presentation space.

Asynchronous Mode

When asynchronous mode is enabled by calling **WinHLLAPIAsync**, the function initiates keystroke intercept and immediately returns control to your Windows HLLAPI application. This frees your application to perform other tasks while waiting for keystrokes.

Because asynchronous mode returns control immediately, you must use Windows version 3.x message notification to determine when keystrokes have occurred. Use the **RegisterWindowsMessage( )** function to register the message “WinHLLAPIAsync”. See **WinHLLAPIAsync** in Chapter 4 for details.

Stop Close Intercept—Function 43

This function stops the application from intercepting close requests from the user. Subsequent close requests are processed normally by the emulator program.

Prerequisite Functions

Start Close Intercept (function 41).

Function Call

WinHLLAPI(STOPCLOSEINTERCEPT,*lpbyString***,***lpwLength***,** *lpwReturnnCode***)**

Call Parameters

Parameter	Description
Data String	One-character short name session ID of the presentation space.
Data Length	NA.
PS Position	NA

Return Codes

Code	Description
WHLLOK	The function was successful.
WHLLNOTCONNECTED	An invalid presentation space was specified, or was not connected for window services.
WHLLPARAMETERERROR	An invalid option was specified.
WHLLNOTAVAILABLE	Start Close Intercept has not been called prior to this function for the specified presentation space.
WHLLSYSERROR	The function failed due to a system error.
WHLLPSENDED	The session stopped.

Remarks

This function is not supported for 5250 emulation.

Stop Host Notification—Function 25

This function disables notifying your Windows HLLAPI application of changes in the Host session presentation space or Operation Information Area (OIA).

Prerequisite Functions

Start Host Notification (function 23).

Function Call

WinHLLAPI(STOPHOSTNOTIFICATION,*lpbyString***,***lpwLength***,** *lpwReturnnCode***)**

Call Parameters

Parameter	Description
Data String	Short name session ID of the desired Host session, or space or null for the current Host session.
Data Length	NA (length of 1 is implied).
PS Position	NA

Return Codes

Code	Description
WHLLOK	Host notification disabled (function successful).
WHLLNOTCONNECTED	The specified Host session is invalid.
WHLLNOTAVAILABLE	Start Host Notification (function 23) has not been called prior to this function for the specified Host session.
WHLLSYSERROR	The function failed due to a system error.

Remarks

Once Host notification has been disabled, Query Host Update (function 24) can no longer determine updates to the Host session, and Host events do not satisfy Pause (function 18).

Stop Keystroke Intercept—Function 53

This function disables the ability of your Windows HLLAPI application to intercept keystrokes.

Prerequisite Functions

Start Keystroke Intercept (function 50).

Function Call

WinHLLAPI(STOPKSINTERCEPT,*lpbyString***,***lpwLength***,**
*lpwReturnnCode***)**

Call Parameters

Parameter	Description
Data String	One byte: short name session ID of the desired Host session, or space or null for the current Host session.
Data Length	NA (length of 1 is implied).
PS Position	NA.

Return Codes

Code	Description
WHLLOK	Keystroke intercept has been enabled.
WHLLNOTCONNECTED	The Host session presentation space is invalid.
WHLLNOTAVAILABLE	Start Keystroke Intercept (function 50) was not called prior to this function call.
WHLLSYSERROR	The function failed due to a system error.

Wait—Function 4

This function determines whether the Host session is in a wait state. If, for some reason, the session is in a wait state, this function causes your Windows HLLAPI application to wait for the specified amount of time to see if the wait condition clears. The amount of time to wait is set by session options with Set Session Parameters (function 9).

Prerequisite Functions

Connect Presentation Space (function 1).

Function Call

WinHLLAPI(WAIT,*lpbyString***,***lpwLength***,***lpwReturnnCode***)**

WinHLLAPIAsync(*hWnd***,WAIT,***lpbyString***,***lpwLength***,***lpwReturnnCode***)**

Call Parameters

Parameter	Description
Data String	NA
Data Length	NA
PS Position	NA

Return Codes

Code	Description
WHLLOK	The keyboard is unlocked and ready for input.
WHLLNOTCONNECTED	Your Windows HLLAPI application is not connected to a valid Host session.
WHLLPSBUSY	Wait function timed out while still in XCLOCK or XSYSTEM for 3270 terminals, or Input Inhibited for 5250 terminals.
WHLLINHIBITED	The keyboard is locked.
WHLLSYSERROR	The function failed due to a system error.
WHLLCANCEL	The asynchronous function was cancelled.

Remarks

Wait can be used to provide other functions, such as Send Key (function 3), enough time to complete or be processed. You can also use Wait to see if the keyboard is inhibited (return code of 4). Be aware, however, that when the return code is 0 (zero), the keyboard is unlocked and Wait has executed successfully, but the original transaction or preceding function may not have finished processing on the Host. If there are keywords or prompts you are expecting, use Search Field (function 30) or Search Presentation Space (function 6) in combination with Wait.

The length of time that this function will wait is affected by the session options TWAIT, LWAIT, and NWAIT. See Set Session Parameters (function 9) for details on these session options.

Although both APIs are supported, you should use **WinHLLAPIAsync** instead of **WinHLLAPI** whenever possible. Note that if NWAIT is specified, the **WinHLLAPIAsync** call will work the same as the **WinHLLAPI** call and not send a message.

Window Status—Function 104

This function allows the application to query or change a session's window size, location, or visible state, or to query a session's window handle or font characteristics.

Prerequisite Functions

Connect Window Services (function 101).

Function Call

WinHLLAPI(WINDOWSTATUS,*lpbyString***,***lpwLength***,**
*lpwReturnnCode***)**

Call Parameters

Parameter	Description
Data String	See the following tables.
Data Length	NA (defaults to 16 or 20, depending on the status request).
PS Position	NA

Data String	Description
Set window status - a 16-byte string with the following format:	
Byte 1	Short name session ID of the desired host session, or space or null for the current host session.
Byte 2	WHLL_WINDOWSTATUS_SET for set status.
Bytes 3-4	An integer containing the set values. The following are valid: u WHLL_WINDOWSTATUS_SIZE. Change the window size (not valid with minimize, maximize, restore, or move). u WHLL_WINDOWSTATUS_MOVE. Change the window x or y position (not valid with minimize, maximize, size, or restore). u WHLL_WINDOWSTATUS_ZORDER. Specifies window z-order placement. u WHLL_WINDOWSTATUS_SHOW. Set the window to visible. u WHLL_WINDOWSTATUS_HIDE. Set the window to invisible. u WHLL_WINDOWSTATUS_ACTIVATE. Activate the window. Use the _ZORDER placement if specified, otherwise set focus to the window and place it in the foreground. u WHLL_WINDOWSTATUS_DEACTIVATE. Deactivate the window. Use the _ZORDER placement if specified, otherwise place it in the background. u WHLL_WINDOWSTATUS_MINIMIZE. Set the window to minimized (not valid with maximize, restore, size, or move).

Data String	Description
Set window status - a 16-byte string with the following format:	
	u WHLL_WINDOWSTATUS_MAXIMIZE. Set the window to maximized (not valid with minimize, restore, size, or move). u WHLL_WINDOWSTATUS_RESTORE. Restore the window (not valid with maximize, minimize, size, or move).
Bytes 5-6	Specifies the x-coordinate of the upper-left corner of the window.
Bytes 7-8	Specifies the y-coordinate of the upper-left corner of the window.
Bytes 9-10	Specifies the width of the window.
Bytes 11-12	Specifies the height of the window.
Bytes 13-16	Specifies the z-order placement of the window (only valid for the set option when the _ZORDER option is specified). Valid values are: u WHLL_WINDOWSTATUS_FRONT. Place window in front. u WHLL_WINDOWSTATUS_BACK. Place window in back.
Query window status - a 16-byte string with the following format:	
Byte 1	Short name session ID of the desired host session, or space or null for the current host session.
Byte 2	WHLL_WINDOWSTATUS_QUERY for query for status.
Bytes 3-4	An integer containing WHLL_WINDOWSTATUS_NULL. The following are possible return values. More than one status is possible. u WHLL_WINDOWSTATUS_SHOW. The window is visible. u WHLL_WINDOWSTATUS_HIDE. The window is invisible. u WHLL_WINDOWSTATUS_ACTIVATE. The window is activated. u WHLL_WINDOWSTATUS_DEACTIVATE. The window is deactivated. u WHLL_WINDOWSTATUS_MINIMIZE. The window is minimized. u WHLL_WINDOWSTATUS_MAXIMIZE. The window is maximized.
Bytes 5-6	Specifies the x-coordinate of the upper-left corner of the window.
Bytes 7-8	Specifies the y-coordinate of the upper-left corner of the window.
Bytes 9-10	Specifies the width of the window.
Bytes 11-12	Specifies the height of the window.
Bytes 13-16	Specifies the z-order placement of the window (only valid for the set option when the _ZORDER option is specified). Valid values are: u WHLL_WINDOWSTATUS_FRONT. Place window in front. u WHLL_WINDOWSTATUS_BACK. Place window in back.

Data String	Description
Query extended window status - a 20-byte string with the following format:	
Byte 1	Short name session ID of the desired host session, or space or null for the current host session.
Byte 2	WHLL_WINDOWSTATUS_EXTQUERY for query extended status.
Bytes 3-4	An integer containing WHLL_WINDOWSTATUS_NULL. The following are possible return values. More than one status is possible. u WHLL_WINDOWSTATUS_SHOW. The window is visible. u WHLL_WINDOWSTATUS_HIDE. The window is invisible. u WHLL_WINDOWSTATUS_ACTIVATE. The window is activated. u WHLL_WINDOWSTATUS_DEACTIVATE. The window is deactivated. u WHLL_WINDOWSTATUS_MINIMIZE. The window is minimized. u WHLL_WINDOWSTATUS_MAXIMIZE. The window is maximized.
Bytes 5-6	Specifies the current font height. The size assumes a fixed-pitch font including any inter-column spacing (this value times the number of displayed columns should equal the width of the presentation space).
Bytes 7-8	Specifies the current font width. The size includes any inter-line spacing (this value times the number of displayed rows should equal the height of the presentation space).
Bytes 9-10	Specifies the distance from the left edge of the window to the first displayed column of the host screen, or zero if the host presentation space exactly fits the window.
Bytes 11-12	Specifies the distance from the top of the window to the first displayed row of the host screen, or zero if the host presentation space exactly fits the window.
Bytes 13-14	Specifies the number of the first visible row of the presentation space. This is normally one unless only a portion of the presentation space is visible in the window.
Bytes 15-16	Specifies the number of the first visible column of the presentation space. This is normally one unless only a portion of the presentation space is visible in the window.
Bytes 17-20	Specifies the window handle of the emulator session. For Win16 handles, only positions 17-18 are used.

Return Codes

Code	Description
WHLLOK	The function was successful.
WHLLNOTCONNECTED	An invalid presentation space was specified, or was not connected for window services.
WHLLPARAMETERERROR	An invalid option was specified.
WHLLSYSERROR	The function failed due to a system error.
WHLLPSENDED	The session stopped.

Remarks

All coordinate positions, screen sizes, offsets, and font sizes are in pixels. This function is not supported for 5250 emulation.
When resizing a window, the requested size and position may be slightly different then what was requested. Follow the set option with a query option to determine the final window position and size.

CHAPTER 4

Extensions for the Windows Environment

This chapter describes API extensions to Windows HLLAPI that allow asynchronous communication. These extensions have been designed for all implementations and versions of the Microsoft Windows graphical environment starting from Microsoft Windows version 3.0. They provide for Windows HLLAPI implementations and applications in 16- and 32-bit operating environments.

Windows HLLAPI allows multithreaded Windows-based processes. A process contains one or more threads of execution. In the non-multithreaded world of the 16-bit Windows environment, a task corresponds to a process with a single thread. All references to threads in this document refer to actual threads in multithreaded Windows environments. In non multithreaded environments, such as the Windows version 3.0 graphical environment, “thread” is synonymous with “process.”

The extensions for the Windows environment included in Windows HLLAPI are provided for maximum Microsoft Windows programming compatibility and optimum application performance. Each of these function calls have corresponding prototypes in the WHLLAPI.H header file, found in Appendix A.

WinHLLAPIAsync( )

This function provides an asynchronous flavor to the following HLLAPI functions: STARTKSINTERCEPT, WAIT, STARTHOSTNOTIFICATION, STARTCLOSEINTERCEPT, SENDFILE, and RECEIVEFILE. You should use **WinHLLAPIAsync()** instead of the blocking versions of these functions.

Syntax

HANDLE WinHLLAPIAsync(*hWnd***,***lpwFunction***,***lpbyString***,***lpwLength***,**
*lpwReturnCode***);**

When the asynchronous operation is complete, the application's window *hWnd* receives the message returned by **RegisterWindowMessage** with "WinHLLAPIAsync" or "WinHLLAPIAsyncFileTransfer" as the input string. For STARTKSINTERCEPT, WAIT, STARTHOSTNOTIFICATION, and STARTCLOSEINTERCEPT, The *wParam* argument contains the asynchronous task handle as returned by the original function call. The high 16 bits of *lParam* contain any error code. The error code may be any error as defined in WHLLAPI.H. An error code of zero indicates successful completion of the asynchronous function. The low 16 bits contains the original function number. For SENDFILE and RECEIVEFILE, the *wParam* and *lParam* contain status information. See the Asynchronous Mode section of Send File and Receive File for details.

Returns

The return value specifies whether the asynchronous resolution request was successful.
It is nonzero if the operation was successful and the actual return value is an asynchronous task handle that can be subsequently used to cancel the asynchronous resolution request if necessary. It is zero if the function failed.
The asynchronous function can be canceled at any time by passing the handle returned by **WinHLLAPIAsync** to **WinHLLAPICancelAsyncRequest()**.

Windows HLLAPI Supplier Notes

The Windows HLLAPI supplier must ensure that messages are successfully posted to the application. If a **PostMessage()** operation fails, the Windows HLLAPI implementation *must* re-post that message.
See also: **WinHLLAPICancelAsyncRequest()**

WinHLLAPICleanup( )

This routine should be called by an application to deregister itself from a Windows HLLAPI implementation.

Syntax **BOOL WinHLLAPICleanup(void)**

Returns The return value indicates whether the deregistration was successful. It is non-zero if the application was successfully deregistered; otherwise it is zero.

Windows HLLAPI Supplier Notes

Use the **WinHLLAPICleanup()** call to indicate deregistration of a Windows HLLAPI application from a Windows HLLAPI implementation. This function can be used, for example, to free up resources allocated to the specific application.

See also: **WinHLLAPIStartup()**

WinHLLAPIIsBlocking( )

This function allows a task to determine if it is executing while waiting for a previous blocking call to complete.

Syntax

BOOL WinHLLAPIIsBlocking(void)

Returns

The return value specifies the outcome of the function. It is nonzero if there is an outstanding blocking call awaiting completion; otherwise it is zero.

Remarks

Although a call issued on a blocking function appears to an application as though it blocks, the WHLLAPI DLL has to relinquish the processor to allow other applications to run. This means that it is possible for the application that issued the blocking call to be re-entered, depending on the message(s) it receives. In this instance, the **WinHLLAPIIsBlocking()** call can be used to determine whether the application task currently has been re-entered while waiting for an outstanding blocking call to complete. Note that Windows HLLAPI prohibits more than one outstanding blocking call per thread.

Windows HLLAPI Supplier Notes

A Windows HLLAPI implementation must prohibit more than one outstanding blocking call per thread.

WinHLLAPICancelAsyncRequest( )

This function cancels an outstanding **WinHLLAPIAsync()**-based request.

Syntax

int WinHLLAPICancelAsyncRequest(HANDLE *hAsyncTaskID*,**WORD** *wFunction***)**

An asynchronous task previously initiated by issuing one of the **WinHLLAPIAsync()** functions can be canceled prior to completion by issuing the **WinHLLAPICancelAsyncRequest()** function and specifying the asynchronous task ID as returned by the initial function in the *hAsyncTaskID* parameter and the WinHLLAPI function number.

Parameter	Type	Description
hAsyncTaskID	**HANDLE**	Specifies the asynchronous task to be canceled.
wFunction	**WORD**	Specifies the function number to be canceled.

Returns

The return value specifies whether the original asynchronous request was canceled. It is zero if the request was canceled; otherwise it is on of the following return codes:

WHLLINVALID	Indicates that the specified asynchronous task ID was invalid.
WHLLALREADY	The asynchronous routine being canceled has already completed.

Remarks

Should an attempt to cancel an existing asynchronous **WinHLLAPIAsync()** routine fail with an error code of WHLLALREADY, it can be for one of 2 reasons. Firstly, the original routine has already completed and the application has dealt with the resultant message. Secondly, the original routine has already completed but the resultant message is still waiting in the application window queue.

See also: **WinHLLAPICancelAsyncRequest()**

WinHLLAPICancelBlockingCall( )

This function cancels any outstanding blocking operation for its thread. Any outstanding blocked call canceled will cause an error code of WHLLCANCEL to be generated. Examples of blocking calls are **WinHLLAPI** with function number set to GETKEY, WAIT, PAUSE, SENDFILE or RECEIVEFILE. You should use **WinHLLAPIAsync()** instead of the blocking versions of these functions.
Under Windows NT, a multi-threaded application may have multiple blocking operations outstanding; but only one per thread. To distinguish between multiple outstanding calls, **WinHLLAPICancelBlockingCall** cancels the outstanding operation on the current (i.e. calling) application thread if one exists; otherwise it fails. By default under Windows NT, WinHLLAPI will suspend the calling application thread while an operation is outstanding. As a result, the thread on which the blocking operation was initiated will not regain control (and hence will not be able to issue a call to **WinHLLAPICancelBlockingCall**) unless a blocking hook is registered for the thread using **WinHLLAPISetBlockingHook**. This condition does not apply to Windows version 3.*x* since applications only have one effective thread and the default blocking hook is registered by default.

Syntax

int WinHLLAPICancelBlockingCall(void)

Returns

The return value indicates whether the cancellation request was successful. It is zero if the operation was successful; otherwise it is one of the following return codes:

WHLLINVALID	Indicates that there is no outstanding blocking call.

See also: **WinHLLAPICancelAsyncRequest()**

WinHLLAPIStartup( )

This function allows an application to specify the version of Windows HLLAPI required and to retrieve details of the specific Windows HLLAPI implementation. This function **MUST** be called by an application before issuing any further Windows HLLAPI calls to register itself with a Windows HLLAPI implementation.

Syntax

int WinHLLAPIStartup(WORD *wVersionRequired***, LPWHLLAPIDATA** *lpData***)**

In order to support future Windows HLLAPI implementations and applications that may have functionality differences from Windows HLLAPI version 1.0, a negotiation takes place in **WinHLLAPIStartup()**. An application passes to **WinHLLAPIStartup()** the Windows HLLAPI version of which it can take advantage. If this version is lower than the lowest version supported by the Windows HLLAPI DLL, then the DLL cannot support the application and the **WinHLLAPIStartup()** call fails. Otherwise, the call succeeds and returns the highest version of Windows HLLAPI supported by the DLL. If this version is lower than the lowest version supported by the application, the application either fails its initialization or attempts to find another Windows HLLAPI DLL on the system.

This negotiation allows both a Windows HLLAPI DLL and a Windows HLLAPI application to support a range of Windows HLLAPI versions. An application can successfully use a DLL if there is any overlap in the versions. The following chart gives examples of how **WinHLLAPIStartup()** works in conjunction with different application and DLL versions:

Details of the actual Windows HLLAPI implementation are described in the WHLLAPIDATA structure defined as follows:

```
typedef struct tagWHLLAPIDATA {
    WORD wVersion;
    char     szDescription[WHLLDESCRIPTION_LEN+1];
} WHLLAPIDATA, * PWHLLAPIDATA, FAR * LPWHLLAPIDATA;
```

App versions	DLL Versions	To WinHLLAPI Startup	From WinHLLAPI Startup	Result
1.0	1.0	1.0	1.0	use 1.0
1.0 2.0	1.0	2.0	1.0	use 1.0
1.0	1.0 2.0	1.0	2.0	use 1.0
1.0	2.0 3.0	1.0	WHLLINVALID	fail
2.0 3.0	1.0	3.0	1.0	app fails
1.0 2.0 3.0	1.0 2.0 3.0	3.0	3.0	use 3.0

Having made its last Windows HLLAPI call, an application should call the **WinHLLAPICleanup()** routine.

Parameter	Type	Description
wVersionRequired	**WORD**	Specifies the version of Windows HLLAPI support required. The high order byte specifies the minor version (revision) number; the low-order byte specifies the major version number.
lpData	Structure	Containing information about the underlying Windows HLLAPI DLL implementation. The first *wVersion* field has the same structure as the *wVersionRequired* parameter and the *szDescription* field contains a string identifying the vendor of the Windows HLLAPI DLL. The description field is only meant to provide a display string for the application and should not be used to programatically distinguish between Windows HLLAPI implementations.

Returns

The return value indicates whether the application was registered successfully and whether the Windows HLLAPI implementation can support the specified version number. It is zero if it was registered successfully and the specified version can be supported; otherwise it is one of the following return codes:

WHLLSYSNOTREADY	Indicates that the underlying network subsystem is not ready for network communication.
WHLLVERNOTSUPPORTED	The version of Windows HLLAPI support requested is not provided by this particular Windows HLLAPI implementation.
WHLLINVALID	The Windows HLLAPI version specified by the application is not supported by this DLL.

Windows HLLAPI Supplier Notes

Each Windows HLLAPI implementation must make a **WinHLLAPIStartup()** call before issuing any other Windows HLLAPI calls. This function can thus be used for initialization purposes.

See also: **WinHLLAPICleanup()**

WinHLLAPISetBlockingHook( )

This function installs a new function which a Windows HLLAPI Implementation should use to implement blocking HLLAPI function calls.
This mechanism is provided to allow a Windows version 3.*x* application to make blocking calls without blocking the rest of the system. By default under Windows NT, blocking calls will suspend the calling application's thread until the request completes. Therefore if a single-threaded application is targeted at both Windows version 3.*x* and Windows NT and relies on this functionality it should register a blocking hook even if the default hook would suffice.

Syntax

FARPROC WinHLLAPISetBlockingHook(FARPROC *lpBlockFunc***)**

Description

A Windows HLLAPI Implementation has a default mechanism by which blocking HLLAPI functions are implemented. This function gives the application the ability to execute its own function at blocking time in place of the default function.

Parameter	Type	Description
lpBlockFunc	**FARPROC**	Specifies the procedure instance address of the blocking function to be installed.

The default blocking function is equivalent to:

```
BOOL DefaultBlockingHook(void) {
     MSG msg;
     /* get the next message if any */
     if( PeekMessage(&msg,0,0,PM_NOREMOVE) ) {
          if ( msg.message == WM_QUIT )
               return FALSE;  // let app process WM_QUIT
          PeekMessage(&msg,0,0,PM_REMOVE);
          TranslateMessage(&msg);
          DispatchMessage(&msg);
     }
     /* TRUE if no WM_QUIT received */
     return TRUE;
}
```

The **WinHLLAPISetBlockingHook** function is provided to support those applications which require more complex message processing - for example, those employing the MDI (multiple document interface) model.
Blocking functions must return FALSE if it receives a WM_QUIT message so WinHLLAPI can return control to the application to process the message and terminate gracefully. Otherwise the function should return TRUE.

Returns

The return value points to the procedure-instance of the previously installed blocking function. The application or library that calls the **SetBlockingHook** function should save this return value so that it can be restored if necessary. (If "nesting" is not important, the application may simply discard the value returned by **WinHLLAPISetBlockingHook** and eventually use **WinHLLAPIUnhookBlockingHook** to restore the default mechanism.)

Windows HLLAPI Supplier Notes

This function must be implemented on a per-thread basis. It thus provides for a particular thread to replace the blocking mechanism without affecting other threads.
See also: **WinHLLAPIUnhookBlockingHook()**

WinHLLAPIUnhookBlockingHook( )

This function removes any previous blocking hook that has been installed and reinstalls the default blocking mechanism.

Syntax **BOOL WinHLLAPIUnhookBlockingHook(void)**

Returns The return value specifies the outcome of the function. It is nonzero if the default mechanism is successfully reinstalled; otherwise it is zero.

See also: **WinHLLAPISetBlockingHook()**

APPENDIX A

WHLLAPI.H - Definitions / Declarations for the Windows HLLAPI Specification

```
/*****************************************************************************\
*                                                                             *
* whllapi.h -   Windows HLLAPI functions, types, and definitions              *
*                                                                             *
*               Version 1.0                                                   *
*                                                                             *
\*****************************************************************************/

/****** Function numbers *********************************************/

#define OEMFUNCTION               0  /* OEM Function */
#define CONNECTPS                 1  /* Connect Presentation Space */
#define DISCONNECTPS              2  /* Disconnect Presentation Space */
#define SENDKEY                   3  /* Send Key */
#define WAIT                      4  /* Wait */
#define COPYPS                    5  /* Copy Presentation Space */
#define SEARCHPS                  6  /* Search Presentation Space */
#define QUERYCURSORLOC            7  /* Query Cursor Location */
#define COPYPSTOSTR               8  /* Copy Presentation Space To String */
#define SETSESSIONPARAMETERS      9  /* Set Session Parameters */
#define QUERYSESSIONS            10  /* Query Sessions */
#define RESERVE                  11  /* Reserve */
#define RELEASE                  12  /* Release */
#define COPYOIA                  13  /* Copy OIA Information */
#define QUERYFIELDATTRIBUTE      14  /* Query Field Attribute */
#define COPYSTRTOPS              15  /* Copy String To Presentation Space */
#define STORAGEMGR               17  /* Storage Manager */
#define PAUSE                    18  /* Pause */
#define QUERYSYSTEM              20  /* Query System */
#define RESETSYSTEM              21  /* Reset System */
#define QUERYSESSIONSTATUS       22  /* Query Session Status */
#define STARTHOSTNOTIFICATION    23  /* Start Host Notification */
#define QUERYHOSTUPDATE          24  /* Query Host Update */
#define STOPHOSTNOTIFICATION     25  /* Stop Host Notification */
#define SEARCHFIELD              30  /* Search Field */
#define FINDFIELDPOSITION        31  /* Find Field Position */
```

```
#define FINDFIELDLENGTH           32  /* Find Field Length */
#define COPYSTRINGTOFIELD         33  /* Copy String To Field */
#define COPYFIELDTOSTRING         34  /* Copy String To Field */
#define SETCURSOR                 40  /* Set Cursor */
#define STARTCLOSEINTERCEPT       41  /* Start Close Intercept */
#define QUERYCLOSEINTERCEPT       42  /* Query Close Intercept */
#define STOPCLOSEINTERCEPT        43  /* Stop Close Intercept */
#define STARTKSINTERCEPT          50  /* Start Keystroke Intercept */
#define GETKEY                    51  /* Get Key */
#define POSTINTERCEPTSTATUS       52  /* Post Intercept Status */
#define STOPKSINTERCEPT           53  /* Stop  Keystroke Intercept */
#define LOCKPSAPI                 60  /* Lock Presentation Space API */
#define LOCKWSAPI                 61  /* Lock Window Services API */
#define SENDFILE                  90  /* Send File */
#define RECEIVEFILE               91  /* Receive File */
#define CONVERT                   99  /* Convert Position or RowCol */
#define CONNECTWINDOWSERVICES    101  /* Connect Window Services */
#define DISCONNECTWINDOWSERVICES 102  /* Disconnect Window Services */
#define QUERYWINDOWCOORDINATES   103  /* Query or Set Window Coordinates */
#define WINDOWSTATUS             104  /* Query or Set Window Status */
#define CHANGEPSNAME             105  /* Change Presentation Space Name */
#define CONNECTSTRFLDS           120  /* Connect Structured Fields */
#define DISCONSTRFLDS            121  /* Disconnect Structured Fields */
#define QUERYCOMMBUFSIZ          122  /* Query Communications Buffer Size */
#define ALLOCCOMMBUFF            123  /* Allocate Communications Buffer */
#define FREECOMMBUFF             124  /* Free Communications Buffer */
#define GETREQUESTCOMP           125  /* Get Request Completion */
#define READSTRFLDS              126  /* Read Structured Fields */
#define WRITESTRFLDS             127  /* Write Structured Fields */

/****** SetSessionParameters values ***************************************/

#define WHLL_SSP_NEWRET      (DWORD)0x00000001
#define WHLL_SSP_OLDRET      (DWORD)0x00000002
#define WHLL_SSP_ATTRB       (DWORD)0x00000004
#define WHLL_SSP_NOATTRB     (DWORD)0x00000008
#define WHLL_SSP_NWAIT       (DWORD)0x00000010
#define WHLL_SSP_LWAIT       (DWORD)0x00000020
#define WHLL_SSP_TWAIT       (DWORD)0x00000040
#define WHLL_SSP_EAB         (DWORD)0x00000080
#define WHLL_SSP_NOEAB       (DWORD)0x00000100
#define WHLL_SSP_AUTORESET   (DWORD)0x00000200
#define WHLL_SSP_NORESET     (DWORD)0x00000400
#define WHLL_SSP_SRCHALL     (DWORD)0x00001000
#define WHLL_SSP_SRCHFROM    (DWORD)0x00002000
#define WHLL_SSP_SRCHFRWD    (DWORD)0x00004000
#define WHLL_SSP_SRCHBKWD    (DWORD)0x00008000
#define WHLL_SSP_FPAUSE      (DWORD)0x00010000
#define WHLL_SSP_IPAUSE      (DWORD)0x00020000
```

```
/****** Convert Row or Column values ************************************/

#define WHLL_CONVERT_POSITION 'P'
#define WHLL_CONVERT_ROW      'R'

/******* Storage Manager Sub-Function values ******************************/

#define WHLL_GETSTORAGE         1
#define WHLL_FREESTORAGE        2
#define WHLL_FREEALLSTORAGE     3
#define WHLL_QUERYFREESTORAGE   4

/****** Change PS Name values ***************************************************/

#define WHLL_CHANGEPSNAME_SET          0x01
#define WHLL_CHANGEPSNAME_RESET             0x02

/****** Window Status values ****************************************************/

#define WHLL_WINDOWSTATUS_SET               0x01
#define WHLL_WINDOWSTATUS_QUERY                  0x02
#define WHLL_WINDOWSTATUS_EXTQUERY          0x03

#define WHLL_WINDOWSTATUS_NULL                   0x0000
#define WHLL_WINDOWSTATUS_SIZE                   0x0001
#define WHLL_WINDOWSTATUS_MOVE                   0x0002
#define WHLL_WINDOWSTATUS_ZORDER            0x0004
#define WHLL_WINDOWSTATUS_SHOW                   0x0008
#define WHLL_WINDOWSTATUS_HIDE                   0x0010
#define WHLL_WINDOWSTATUS_ACTIVATE          0x0080
#define WHLL_WINDOWSTATUS_DEACTIVATE        0x0100
#define WHLL_WINDOWSTATUS_MINIMIZE          0x0400
#define WHLL_WINDOWSTATUS_MAXIMIZE          0x0800
#define WHLL_WINDOWSTATUS_RESTORE           0x1000

#define WHLL_WINDOWSTATUS_FRONT         (DWORD)0x00000003
#define WHLL_WINDOWSTATUS_BACK          (DWORD)0x00000004

/****** Lock API values *********************************************************/

#define WHLL_LOCKAPI_LOCK           'L'
#define WHLL_LOCKAPI_UNLOCK              'U'
#define WHLL_LOCKAPI_RETURN              'R'
#define WHLL_LOCKAPI_QUEUE               'Q'
```

```
/****** Windows HLLAPI Return Codes ***************************************/

#define WHLLOK                    0   /* Successful */
#define WHLLNOTCONNECTED          1   /* Not Connected To Presentation Space */
#define WHLLBLOCKNOTAVAIL         1   /* Requested size is not available */
#define WHLLPARAMETERERROR        2   /* Parameter Error/Invalid Function */
#define WHLLBLOCKIDINVALID        2   /* Invalid Block ID was specified */
#define WHLLFTXCOMPLETE           3   /* File Transfer Complete */
#define WHLLFTXSEGMENTED          4   /* File Transfer Complete / segmented */
#define WHLLPSBUSY                4   /* Presentation Space is Busy */
#define WHLLINHIBITED             5   /* Inhibited/Keyboard Locked */
#define WHLLTRUNCATED             6   /* Data Truncated */
#define WHLLPOSITIONERROR         7   /* Invalid Presentation Space Position */
#define WHLLNOTAVAILABLE          8   /* Unavailable Operation */
#define WHLLSYSERROR              9   /* System Error */
#define WHLLNOTSUPPORTED          10  /* Function Not Supported */
#define WHLLUNAVAILABLE           11  /* Resource is unavailable */
#define WHLLPSENDED               12  /* The session was stopped */
#define WHLLUNDEFINEDKEY          20  /* Undefined Key Combination */
#define WHLLOIAUPDATE             21  /* OIA Updated */
#define WHLLPSUPDATE              22  /* PS Updated */
#define WHLLBOTHUPDATE            23  /* Both PS And OIA Updated */
#define WHLLNOFIELD               24  /* No Such Field Found */
#define WHLLNOKEYSTROKES          25  /* No Keystrokes are available */
#define WHLLPSCHANGED             26  /* PS or OIA changed */
#define WHLLFTXABORTED            27  /* File transfer aborted */
#define WHLLZEROLENFIELD          28  /* Field length is zero */
#define WHLLKEYOVERFLOW           31  /* Keystroke overflow */
#define WHLLSFACONN               32  /* Other application already connected */
#define WHLLTRANCANCLI            34  /* Message sent inbound to host cancelled */
#define WHLLTRANCANCL             35  /* Outbound trans from host cancelled */
#define WHLLHOSTCLOST             36  /* Contact with host was lost */
#define WHLLOKDISABLED            37  /* The function was successful */
#define WHLLNOTCOMPLETE           38  /* The requested fn was not completed */
#define WHLLSFDDM                 39  /* One DDM session already connected */
#define WHLLSFDPEND               40  /* Disconnected w async requests pending */
#define WHLLBUFFINUSE             41  /* Specified buffer currently in use */
#define WHLLNOMATCH               42  /* No matching request found */
#define WHLLLOCKERROR             43  /* API already locked or unlocked */

#define WHLLINVALIDFUNCTIONNUM   301  /* Invalid function number */
#define WHLLFILENOTFOUND         302  /* File Not Found */
#define WHLLACCESSDENIED         305  /* Access Denied */
#define WHLLMEMORY               308  /* Insufficient Memory */
#define WHLLINVALIDENVIRONMENT   310  /* Invalid environment */
#define WHLLINVALIDFORMAT        311  /* Invalid format */

#define WHLLINVALIDPSID         9998  /* Invalid Presentation Space ID */
#define WHLLINVALIDRC           9999  /* Invalid Row or Column Code */
```

```
/****** Windows HLLAPI Extentions Return Codes ****************************/

#define WHLLALREADY         0xF000   /* An async call is already outstanding */
#define WHLLINVALID         0xF001   /* Async Task Id is invalid */
#define WHLLCANCEL          0xF002   /* Blocking call was cancelled */
#define WHLLSYSNOTREADY     0xF003   /* Underlying subsystem not started */
#define WHLLVERNOTSUPPORTED 0xF004   /* Application version not supported */

/****** Windows HLLAPI structure ******************************************/

#define WHLLDESCRIPTION_LEN   127

typdef struct tagWHLLAPIDATA {
    WORDwVersion;
    charszDescription[WHLLDESCRIPTION_LEN+1];
} WHLLAPIDATA, * PWHLLAPIDATA, FAR * LPWHLLAPIDATA;

/****** Windows HLLAPI Function Prototypes ********************************/

extern WORD WINAPI WinHLLAPI(lpWord, lpStr, lpWord, lpWord);
extern HANDLE WINAPI WinHLLAPIAsync(HWND, LPCSV);
extern BOOL WINAPI WinHLLAPICleanup(void);
extern BOOL WINAPI WinHLLAPIIsBlocking(void);
extern int WINAPI WinHLLAPICancelAsyncRequest(HANDLE, WORD);
extern int WINAPI WinHLLAPICancelBlockingCall(void);
extern int WINAPI WinHLLAPIStartup(WORD, LPWHLLAPIDATA);
extern FARPROC WINAPI WinHLLAPISetBlockingHook(FARnPROC);
extern BOOL WINAPI WinHLLAPIUnhookBlockingHook(void);
```

A P P E N D I X B

Attributes

This appendix contains the following tables:

- 3270 and 5250 Character Attributes.
- 3270 and 5250 Character Color Attributes.
- 3270 and 5250 Field Attributes.

Note The attribute bit positions are in IBM format. The leftmost bit in the byte is 0.

Character Attributes

3270 character attributes

Bit Position	Meaning
0-1	Character highlighting 00 = Normal 10 = Reverse video 01 = Blink 11 = Underline
2-4	Character color (color remap may override this definition) 000 = Default 100 = Green 001 = Blue 101 = Turquoise 010 = Red 110 = Yellow 011 = Pink 111 = White
5-7	Reserved (not used)

5250 Character Attributes

Bit Position	Meaning
0	Reverse image 0 = Normal 1 = Reverse
1	Underscore 0 = None 1 = Underscore
2	Blink 0 = None 1 = Blink
3	Column separators 0 = None 1 = Column separators
4-7	Reserved (not used)

Character Color Attributes

Bit Position	Meaning	
0-3	Background character colors	
	0000 = Black	0100 = Red
	0001 = Blue	0101 = Magenta
	0010 = Green	0110 = Brown
	0011 = Cyan	0111 = White
4-7	Foreground character colors	
	0000 = Black	1000 = Gray
	0001 = Blue	1001 = Light blue
	0010 = Green	1010 = Light green
	0011 = Cyan	1011 = Light cyan
	0100 = Red	1100 = Light red
	0101 = Magenta	1101 = Light magenta
	0110 = Brown	1110 = Yellow
	0111 = White	1111 = High intensity white

Field Attributes

3270 field attributes

Bit Position	Meaning
0-1	Both set to 1 (field attribute byte)
2	Unprotected/protected
	0 = Unprotected data field
	1 = Protected data field
3	Alpha/numeric
	0 = Alphanumeric data
	1 = Numeric data only
4-5	I/SPD
	00 = Normal intensity, pen not detectable
	01 = Normal intensity, pen detectable
	10 = High intensity, pen detectable
	11 = Non-display, pen not detectable
6	Reserved
7	MDT (Modified Data Tag)
	0 = Field has not been modified
	1 = Field has been modified

5250 field attributes

Bit Position	Meaning
0	Field attribute flag
	0=Not a field attribute
	1=Field attribute byte
1	Visibility
	0 = Non-display
	1 = Display
2	Unprotected/protected
	0 = Unprotected data field
	1 = Protected data field
3	Intensity
	0 = Normal
	1 = High
4-6	Field Type
	000 =Alphanumeric: all characters allowed
	001 = Alphabetic only
	010 = Numeric shift: automatic shift for numerics
	011 = Numeric only
	100=Reserved
	101=Digits:
	110=Magnetic stripe reader data only
	111=Signed Numeric
7	MDT
	0 = Field has not been modified
	1 = Field has been modified

APPENDIX C

Extended Windows HLLAPI Functions

This appendix lists the WinHLLAPI functions defined in release 1.0 of IBM Extended Services for OS/2 EHLLAPI Programming Reference but not required for WinHLLAPI compliance.

Allocate Communications Buffer—Function 123

This function allows the application to obtain exclusive control of a memory buffer to be used for read and write structured field requests. A buffer address must be passed on to the functions that read and write the structured field requests.

Prerequisite Functions

None.

Function Call

WinHLLAPI(ALLOCCOMMBUFF,*lpbyString***,***lpwLength***,**
*lpwReturnnCode***)**

Call Parameters

Parameter	Description
Data String	A 6-byte string with the following format: Byte 1-2 16-bit buffer length requested. (0<size<(64K-8)) or (0<size<X'FFF8') Byte 3-6 Reserved.
Data Length	Must be specified.
PS Position	NA

Return Parameters

Parameter	Description
Data String	A 6-byte string with the following format: Byte 1-2 16-bit buffer length requested. Byte 3-6 32-bit address of the allocated buffer.
Data Length	NA (length of 8 is implied).
PS Position	NA.

Return Codes

Code	Description
WHLLOK	The function was successful.
WHLLNOTCONNECTED	An invalid presentation space was specified, or was not connected for window services.
WHLLPARAMETERERROR	An invalid option was specified.
WHLLSYSERROR	The function failed due to a system error.
WHLLUNAVAILABLE	The requested resource is not available.

Remarks

The buffer address is placed in the returned parameter string. The requested buffer size, from 1 byte to 64K minus 8 bytes, is also in the parameter string. See the description of Query Communications Buffer Size (122) for information regarding the size of the buffer.
Buffers obtained with this function cannot be shared among different processes. Applications that attempt to share these buffers will experience unpredictable results.
Your Windows HLLAPI application must issue a Free Communications Buffer (124) function to free the allocated memory for use by other programs.
The Reset System (21) function call frees any buffers allocated by this function.

Note No more than 10 buffers may be allocated to an application at one time.

When this limit is reached, additional requests to WinHLLAPI will return an 11, indicating that the resource is unavailable.

Connect Structured Fields—Function 120

This function allows an application to establish a connection with a Host session.

Prerequisite Functions

None

Function Call

WinHLLAPI(CONNECTSTRFLDS,*lpbyString***,***lpwLength***,** *lpwReturnnCode***)**

Call Parameters

Parameter	Description	
Data String	An 11-byte string for returned semaphore address. The first byte is a short name session ID of the session to query, or space or null for the current session. Bytes 2-5 are the address of the query reply data buffer.	
	Byte 1	Short name session ID, or space or null for the current session.
	Byte 2-5	Four byte address of the query reply data buffer.
	Bytes 6-11	Reserved for return parameters.
Data Length	Must be specified.	
PS Position	NA	

Return Parameters

Parameter	Description	
Data String	An 11-byte string with the following format:	
	Byte 1	Short name session ID.
	Bytes 2-5	4-byte address of the query reply data buffer.
	Bytes 6-7	16-bit value which represents the destination/origin ID returned to the application by the emulator.
	Bytes 8-11	Address of the semaphore with connection status.

Return Codes

Code	Description
WHLLOK	The function was successful.
WHLLNOTCONNECTED	An invalid presentation space was specified.
WHLLPARAMETERERROR	An invalid option was specified.
WHLLSYSERROR	The function failed due to a system error.
WHLLNOTSUPPORTED	The function is not supported by the emulation program.
WHLLSFACONN	The function failed because another application is already connected to this session.
WHLLSFDDM	The function failed because a DDM session is already connected to this session.

Remarks

Windows HLLAPI scans the query reply buffers for the destination/origin ID (DOID) self-defining parameter (SDP) to obtain the contents of the DOID field of the query reply (that the workstation must supply). A value of X'0000' will cause the emulator to assign a DOID to the workstation application and **WinHLLAPI** will fill in the DOID field of the query reply with the assigned ID. If the value specified is non-zero, the emulator will assign the specified value as the workstation application's DOID, assuming that the ID has not been previously assigned. If the specified DOID is already in use, a return code of 2 will be returned by **WinHLLAPI**.

The application must build the query reply data structures within the application's private memory space. See Appendix D, "Query Reply Data Structures for Windows HLLAPI" for detailed information about structured field usage for the query reply data structures that are supported by **WinHLLAPI**. The 2-byte length field at the beginning of each query reply must not be byte-reversed by the application.

Cursory checking is performed on the query data reply (only the ID and the length of the structure are checked for validity).

The semaphore determines if the state of the structured field connection is set (disabled) or clear (enabled). If the emulator, for example, is in a state that allows processing of a structured field, the semaphore will be clear. If the emulator cannot currently process a structured field, the semaphore will be set. Be sure to check the status of the structured field semaphore before attempting a Read Structured Field (126) or a Write Structured Field (127) function call.

The semaphore is set during the connect process because the emulator is in an inbound disabled state. The semaphore is cleared for the first time when outbound data destined for the connecting DOID is received by the emulator. Because the emulator is in an inbound disabled state, a host application cannot be started via a Write Structured Fields (127) function call. The host application must be started manually, or by issuing a Send Key (3).
Only one DDM base-type connection is allowed, per host session. If the DDM connection supports SDP for the DOID, multiple connections are allowed.
If return code RC=32 or RC=39 is received, an application is already connected to the selected session, and use of that presentation space should be very carefully approached. Otherwise, conflicts with File Transfer or other Windows HLLAPI applications may occur.

Note Structured fields are not supported by the COBOL programming language due to memory access problems inherent to the language.

Disconnect Structured Fields—Function 121

This function drops the connection between the Windows HLLAPI application and the specified session.

Prerequisite Functions

Connect Structured Fields (function 120).

Function Call

WinHLLAPI(DISCONSTRFLDS,*lpbyString***,***lpwLength***,***lpwReturnnCode***)**

Call Parameters

Parameter	Description	
Data String	A 3-byte string with the following format:	
	Byte 1	Short name session ID.
	Bytes 2-3	A 16-bit value which represents the destination/origin ID returned to the application by the Connect Structured Fields (120) function.
Data Length	Must be user-specified.	
PS Position	NA.	

Return Codes

Code	Description
WHLLOK	The function was successful.
WHLLNOTCONNECTED	An invalid presentation space was specified, or was not connected for structured field access.
WHLLPARAMETERERROR	An invalid parameter was specified.
WHLLSYSERROR	The function failed due to a system error.
WHLLSFDPEND	The session was disconnected with asynchronous requests pending.

Remarks

When a Disconnect Structured Fields (121) is called, any active asynchronous Read Structured Fields (126) or Write Structured Fields (127) function requests are returned if the application issues the Get Request Completion (125) function call. Use the asynchronous form of this function when cleaning up after issuing a Disconnect call.

Before exiting the application, you should request the Disconnect Structured Fields (121) function for all emulation sessions that have been connected to using the Connect Structured Fields (120) function. If the application exits with outstanding requests for structured field connections, the those outstanding requests are cancelled. The Reset System (21) function also causes any outstanding requests to be cancelled before disconnecting from structured fields.

Any outstanding asynchronous requests that have not been retrieved by the application using the Get Request Completion (125) function are cleared by the Reset System (21) function, or when **WinHLLAPI** is initialized again.

Note Structured fields are not supported by the COBOL programming language due to memory access problems inherent to the language.

Free Communications Buffer—Function 124

This function allows the application to release exclusive control of a buffer that is no longer required by the application.

Prerequisite Functions

Allocate Communications Buffer (123).

Function Call

WinHLLAPI(FREECOMMBUFF,*lpbyString*,*lpwLength*,*lpwReturnnCode***)**

Call Parameters

Parameter	Description	
Data String	A 6-byte string with the following format:	
	Byte 1-2	16-bit length of the buffer to be freed. If the value of the length specified is 0, the entire buffer is freed.
	Byte 3-6	32-bit address of the buffer obtained from call to Allocate Communications Buffer (123).
Data Length	Must be specified.	
PS Position	NA	

Return Code

Code	Description
WHLLOK	The function was successful.
WHLLNOTCONNECTED	An invalid presentation space was specified, or was not connected.
WHLLPARAMETERERROR	An invalid option was specified.
WHLLSYSERROR	The function failed due to a system error.
WHLLBUFFINUSE	The specified buffer is currently in use.

Remarks

If the application attempts to free a buffer in which the buffer address plus the buffer length overlaps a buffer currently in use, the request is denied and the return code value of 41 (requested buffer in use) is returned. If the application attempts to free an entire selector that contains a buffer in use, the request is also denied and the return code value 41 is returned to the application.

Before exiting an application, you should issue the Free Communications Buffer function call for all communications buffers that have been allocated using the Allocate Communications Buffer (123) function. If the application exits without freeing the buffers, WinHLLAPI will free them when the application exits.

Buffers can also be freed by the Reset System (21) function.

Get Request Completion—Function 125

This function allows the application to determine the status of a previous asynchronous function request issued to WinHLLAPI, and obtains the function parameter list before using the data string again.

Prerequisite Functions

Connect Structured Fields (120) and either Read Structured Fields (126) or Write Structured Fields (127).

Function Call

WinHLLAPI(GETREQUESTCOMP,*lpbyString***,***lpwLength***,** *lpwReturnnCode***)**

Call Parameters

Parameter	Description	
Data String	A 14-byte string with the following format:	
	Byte 1	A 1-character session short name.
	Byte 2	One of the following: N (no wait) W (wait)
	Bytes 3-4	A 16-bit word (2 bytes) into which the function request ID has been placed.
	Bytes 5-14	Reserved for returned parameters
Data Length	NA (defaults to 14).	
PS Position	NA.	

Return Parameters

Parameter	Description	
Data String	If the return code from this function is 0:	
	Bytes 5-6	Two bytes containing the function code of the completed async function.
	Bytes 7-10	Four bytes containing the address of the data string of the completed async function call. The application must not reuse the data string until the request has completed.
	Bytes 11-12	Two bytes containing the length of the data string of the completed async function.
	Bytes 13-14	Two bytes containing the return code of the completed async function.

Return Codes

Code	Description
WHLLOK	The function was successful.
WHLLNOTCONNECTED	An invalid presentation space was specified.
WHLLPARAMETERERROR	An invalid option was specified.
WHLLSYSERROR	The function failed due to a system error.
WHLLNOTCOMPLETE	The requested function was not completed.
WHLLNOMATCH	A matching request was not found.

The difference between returns of WHLLNOTCOMPLETE and WHLLNOMATCH:

WHLLNOTCOMPLETE

If a specific Request ID and session were requested, the session and the ID were found but the request is pending (not yet in a completed state).

If a zero Request ID and specific session were requested, the specified session has pending requests that were not satisfied (completed).

If a Request ID and a blank session were requested, pending requests were found, but none were satisfied.

WHLLNOMATCH

If a specific Request ID and session were requested, the specific ID was *not* found in either a pending or completed state.

If a zero Request ID and specific session were requested, the specified session has *no* pending or completed requests.

If a Request ID and a blank session were requested, *no* pending or completed requests were found.

Remarks

This function is valid only if the user specified asynchronous completion (A) on a previous function call such as Read Structured Fields (126) or Write Structured Fields (127).
Each asynchronous request that requires the Get Request Completion (125) function returns a unique ID from the asynchronous request. The application must save this ID. This ID is the identification used by the Get Request Completion (125) function to identify the request.

The user specifies whether the application can query of wait for one of the following:

- A specific asynchronous function request by supplying the Request ID of that function and a non-blank session short name.
- The first completed asynchronous function request by supplying a Request ID of 0x0000 and a blank session short name.
- The first completed asynchronous function request for a specified session by supplying a Request ID of 0x0000 and a non-blank session short name.
- The Get Request Completion (125) function behaves differently depending upon the second character of the parameter string, which is one of the following:

N (no wait)

If a specific Request ID was supplied and the function completed, control is returned to the application with a return code of zero and a completed data string as defined previously. If a Request ID of 0x0000 was supplied and any eligible asynchronous function has completed, control is returned to the application with a return code of zero and a completed data string as defined previously. If a function has not completed, control is returned to the calling application with a non-zero return code.

W (wait)

If a specific Request ID was supplied and the function has completed:

- The semaphore is cleared

Control is returned to the application with a return code of zero and a completed data string as defined previously

If a Request ID of zero was supplied any eligible function has completed:

- The semaphore is cleared

Control is returned to the application with a return code of zero and a completed data string as defined previously

If a function has not completed, the call waits until a function completes before returning to the application. When it returns, the return code is zero and the data string is completed.

If a nonzero Request ID is supplied, this function checks for the completion of only the function associated with the ID.
If the return code is zero, the application should check the returned data string for information pertaining to the completion of the requested asynchronous function.

Note The communications subsystem allows for a maximum of 20 asynchronous requests per application to be outstanding. A return code for unavailable resources (RC = 11) is returned if more than 20 asynchronous requests are attempted.

Lock Presentation Space API—Function 60

This function allows the application to obtain or release exclusive control of the presentation space.

Prerequisite Functions

Connect Presentation Space (function 1).

Function Call

WinHLLAPI(LOCKPSAPI,*lpbyString***,***lpwLength***,***lpwReturnnCode***)**

Call Parameters

Parameter	Description
Data String	Locking parameters - a 3-byte string with the following format: Byte 1 Short name session ID, or space or null for the current session. Byte 2 One of the following: u WHLL_LOCKAPI_LOCK to lock the API. u WHLL_LOCKAPI_UNLOCK to unlock the API. Byte 3 One of the following: u WHLL_LOCKAPI_RETURN to return if the API is already locked. u WHLL_LOCKAPI_QUEUE to queue the lock request if the API is already locked.
Data Length	Must be specified (normally 3).
PS Position	NA.

Return Codes

Code	Description
WHLLOK	The function was successful.
WHLLNOTCONNECTED	An invalid presentation space was specified, or was not connected.
WHLLPARAMETERERROR	An invalid option was specified.
WHLLSYSERROR	The function failed due to a system error.
WHLLNOTSUPPORTED	The function was not supported by the emulation program.
WHLLPSENDED	The session stopped.
WHLLLOCKERROR	If _LOCK, the API was already locked. If _UNLOCK, the API was not locked.

Remarks

If the API is locked, the WinHLLAPI functions are rejected until the API is unlocked by using the _UNLOCK option, or by disconnecting or resetting the presentation space.

Lock Window Services API—Function 61

This function allows the application to obtain or release exclusive control of the presentation space window services.

Prerequisite Functions

Connect Window Services (function 101).

Function Call

WinHLLAPI(LOCKWSAPI,*lpbyString***,***lpwLength***,***lpwReturnnCode***)**

Call Parameters

Parameter	Description
Data String	Locking parameters - a 3-byte string with the following format: Byte 1 Short name session ID, or space or null for the current session. Byte 2 One of the following: u WHLL_LOCKAPI_LOCK to lock the API. u WHLL_LOCKAPI_UNLOCK to unlock the API. Byte 3 One of the following: u WHLL_LOCKAPI_RETURN to return if the API is already locked. u WHLL_LOCKAPI_QUEUE to queue the lock request if the API is already locked.
Data Length	Must be specified (normally 3).
PS Position	NA.

Return Codes

Code	Description
WHLLOK	The function was successful.
WHLLNOTCONNECTED	An invalid presentation space was specified, or was not connected for window services.
WHLLPARAMETERERROR	An invalid option was specified.
WHLLSYSERROR	The function failed due to a system error.
WHLLNOTSUPPORTED	The function was not supported by the emulation program.
WHLLPSENDED	The session stopped.
WHLLLOCKERROR	If _LOCK, the API was already locked. If _UNLOCK, the API was not locked.

Remarks

If the API is locked, Window Services functions are rejected until the API is unlocked by using the UNLOCK option, or by disconnecting or resetting the presentation space.

Query Communication Buffer Size—Function 122

This function allows the application to determine the maximum and optimum inbound and outbound buffer size supported by the communications engine. These buffer sizes are to be used with the Allocate Communications Buffer (123) function to optimize the performance of the structured field functions.

Prerequisite Functions

None.

Function Call

WinHLLAPI(QUERYCOMMBUFSIZ,*lpbyString***,***lpwLength***,** *lpwReturnnCode***)**

Call Parameters

Parameter	Description
Data String	A 9-byte string with the following format: Byte 1 Short name session ID. Bytes 2-9 Reserved.
Data Length	Must be specified.
PS Position	NA.

Return Parameters

Parameter	Description
Data String	A 9-byte string with the following format: Byte 1 Short name session ID. Bytes 2-3 16-bit value indicating optimum inbound buffer size. Bytes 4-5 16-bit value indicating maximum inbound buffer size. Bytes 6-7 16-bit value indicating optimum outbound buffer size. Bytes 8-9 16-bit value indicating maximum outbound buffer size.

Return Codes

Code	Description
WHLLOK	The function was successful.
WHLLNOTCONNECTED	An invalid presentation space was specified, or was not connected for window services.
WHLLPARAMETERERROR	An invalid option was specified.
WHLLSYSERROR	The function failed due to a system error.
WHLLNOTSUPPORTED	The function is not supported by the emulation program.

Remarks

The buffer sizes that are returned represent the record sizes transmitted across the communications medium. For a DDM connection, the 8-byte header supplied in the Read and Write structured fields data buffer is stripped off and 1 byte containing the structured field AID value is prefixed. The application should compare the size of the actual data in the data buffer (which does not include the 8-byte header) to the buffer sizes returned by the Query Communications Buffer Size function minus 1

byte. For destination/origin connections, the 8-byte header supplied in the Read or Write structured fields data buffer is stripped and 9 bytes are then prefixed to the data. The application should compare the size of the actual data in the data buffer (not including the 8-byte header) to the buffer size returned from the Query Communications Buffer Size (122) function minus 9 bytes.

The maximum buffer sizes represent the maximum number of bytes supported by the PS hardware, and the maximum number of bytes supported by the emulator. The application may use the maximum buffer size **only** if the host system is also configured to accept that size.

The optimum buffer sizes represent the optimal number of bytes supported by both the PC hardware, and the emulator.

If the network configuration sets transmission limits smaller than the optimum buffer size values, the Query Communications Buffer Size (122) call reflects the data transfer buffer size from the current configuration profile.

Read Structured Fields—Function 126

This function receives structured field data from the host application.
If the call specifies asynchronous (A), the application receives control immediately after the call, even if host data is not available. If the call specifies synchronous (S), **WinHLLAPI** waits for host data to become available before returning control to the application.
The application provides the buffer address in which data from the host is to be placed. The buffer must be obtained using the Allocate Communications Buffer (123) function call.

Prerequisite Functions

Connect Structured Fields (function 120).
Allocate Communications Buffer (function 123).

Function Call

WinHLLAPI(READSTRFLDS,*lpbyString***,***lpwLength***,***lpwReturnnCode***)**

Call Parameters

Parameter	Description	
Data String	An 8-byte string for synchronous, or a 14-byte string for asynchronous in the following format:	
	Byte 1	A 1-character session short name.
	Byte 2	A 1-character specifying the control option S (synchronous control) - control is not returned to the application until the read is satisfied. A (asynchronous control) - control is returned immediately to the application.
	Bytes 3-4	The 16-bit word unique destination/origin ID returned by the Connect Structured Fields (120) function call.
	Bytes 5-8	The 4-byte value of the buffer address into which the data is to be read. The buffer must be obtained using the Allocate Communications Buffer (123) function call.
Data Length	Must be 8 or 14.	
PS Position	NA.	

Return Parameters

Parameter	Description	
Data String	When the A (asynchronous) control option is specified and the request is successfully completed, the following are returned:	
	Bytes 9-10	A 16-bit value representing the destination/origin ID returned to the application by the emulator. This function request ID is used by the Get Request Completion (125) function to determine the status of this function call.
	Bytes 11-14	A 4-byte value in which the semaphore address is returned by WinHLLAPI. The application may wait upon this semaphore. When the semaphore is cleared, the application must issue the Get Request Completion (125) function call.

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

Note There is no returned data string for the S (synchronous) control option.

Return Codes

Code	Description
WHLLOK	The function was successful.
WHLLNOTCONNECTED	An invalid presentation space was specified, or was not connected or the DOID was incorrect.
WHLLPARAMETERERROR	An invalid option was specified.
WHLLSYSERROR	The function failed due to a system error.
WHLLUNAVAILABLE	The requested resource was not available.
WHLLTRANCANCL	An outbound transmission from the host was canceled.
WHLLHOSTCLOST	Contact with the host was lost.
WHLLOKDISABLED	The function was successful.

Warning The host inbound transmission is disabled.

The application must correct the situation if one of the following return codes is specified:

WHLLTRANCANCL	Is returned if the first Read Structured Fields (126) or Write Structured Fields (127) is requested after an outbound transmission from the host is canceled.
WHLLHOSTCLOST	Which requires the application to disconnect from the communications subsystem and reconnect to establish communications with the host again.
WHLLOKDISABLED	Which is returned if the host is inbound disabled.

Remarks

When the call to Read Structured Fields (126) is complete, the Read Buffer, whose address was specified in positions 5-8 of the data string, will contain the structured fields received from the host application.

The format of the Read Buffer is as follows:

Position	Meaning
Bytes 0-1	A 16-bit value - 0x0000.
Bytes 2-3	A 16-bit value which contains the Message length (*m*), which is the number of bytes of data in the message not including the 8-byte message header. This value is returned by the subsystem.
Bytes 4-5	A 16-bit value which contains the Buffer Size (*n*), which is the supplied length of the data buffer not including the 8-byte message header.
Bytes 6-7	A 16-bit value - 0xC000
Bytes 8-9	A 16-bit value which contains the Length of the first or only structured field message (not byte-reversed).
Byte 10	First non-length byte of the structured field message.
Byte *m*	Last byte in the structured field message.

Bytes 0-7 are the buffer header, which is passed to and used by the communications subsystem. The application must prepare the buffer header before using it in the structured fields call. The word at position 0 must be set to a value of zero. The length of the buffer, requested with the Allocate Communications Buffer (123) function, must be in the word at position 4. The word at position 6 must be set to 0xC000.

Bytes 8-*m* are where the structured field messages are returned. The following occurs when the call is returned:

- The word at position 2 contains the length (8-*m*) of the structured field messages.
- The word at position 8 contains the length of the first structured field message.

Bytes 10-*m* contain the actual data of the structured field message.

Synchronous Requests

When Read Structured Fields (126) is requested synchronously (the S option in the data string), control is returned to the application only after the request is satisfied. The application can assume:

- The return code is correct.
- The data in the communications buffer (read buffer) is correct.
- The host is no longer processing the Read Structured Fields (126) request.

Asynchronous Requests

When Read Structured Fields (126) is requested asynchronously (the A option in the data string), the application cannot assume:

- The return code is correct.
- The data in the communications buffer (read buffer) is correct.
- The host is no longer processing the Read Structured Fields (126) request.

When requested asynchronously, WinHLLAPI returns the following:

- A 16-bit Request ID in positions 2-3 of the data string.
- The address of a semaphore in positions 4 - 7 of the data string.

These are used to complete the asynchronous Read Structured Fields (126) call.

The following steps must be completed to determine the outcome of an asynchronous Read Structured Fields (126) function call:

- If the WinHLLAPI return code is not zero, the request failed. No asynchronous request has been made. The application must take appropriate actions before attempting the call again.
- If the return code is zero, the application should wait until the semaphore is cleared by using the Get Request Completion (125) function. The semaphore should not be freed (this done by the Get Request Completion (125) function) and should not be reused. The semaphore is only valid for the duration of the Read Structured Fields (126) function call through the completion of the Get Request Completion (125) function call.
- Once the semaphore is cleared, use the returned 16-bit Request ID as the Request ID parameter in a call to the Get Request Completion (125) function. The data string returned from the Get Request Completion (125) function call contains the final return code of the Read Structured Fields (126) function call.

Note The communications subsystem allows for a maximum of 20 asynchronous requests per application to be outstanding. A return code for unavailable resources (RC= 11) is returned if more than 20 asynchronous requests are attempted.

Note Structured fields are not supported by the COBOL programming language due to memory access problems inherent to the language.

Storage Manager—Function 17

This function allows your application elementary control of blocks of memory for use with the Windows HLLAPI function calls.
The Storage Manager (17) function allows easy migration of existing applications (typically BASIC interpreter applications) that use Storage Manager (17) functions. The new BASIC applications can use this function, but are not required to. The other supported languages may also use this function.
There are four available sub-functions to the Storage Manager (17) function:

- Get Storage
- Free Storage
- Free All Storage
- Query Free Storage

Each of the sub-functions has supplied parameters and returned parameters, and generates a set of possible return codes. These sub-functions are discussed in detail in the following pages.
WinHLLAPI returns a return code of WHLLPARAMETERERROR for an invalid sub-function number. The sub-functions are identified to WinHLLAPI by the sub-function number being placed in the *PS Position* calling parameter.
Storage Manager (17) may allocate blocks from 16 bytes to 64 Kbytes in size. The Storage Manager (17) function does ***not*** allocate shared memory.
WinHLLAPI lists the results of the Storage Manager (17) function and places them into a table. Once a request to Get Storage is placed into the table, WinHLLAPI checks the table for free bytes to satisfy the current request. If there is sufficient storage, the free block is marked allocated and is given to the user. If there is not sufficient storage, the user should take what steps are necessary to allocate the memory via normal operating system calls.
When a Free Storage call is made, the specified block is then marked as free in the table.
If a Free All Storage call is made, all blocks in the table are marked as free, and no more use may be made by the application of blocks previously acquired from Get Storage.
A Query Free Storage call returns the size of the single largest area that is currently available.

Prerequisite Functions

None.

Get Storage

The Get Storage sub-function allocates a block of storage to be used by the calling Windows HLLAPI application.

Function Call

WinHLLAPI(STORAGEMGR,*lpbyString***,***lpwLength***,GETSTORAGE)**

Call Parameters

Parameter	Description
Data String	A 4-byte string.
Data Length	Size (in bytes) of the requested storage area.
PS Position	01 (**GETSTORAGE**)

Return Parameters

Parameter	Description
Data String	The storage address is expressed as two binary words: offset and selector. The offset first, then the selector.
Data Length	Storage Block ID of the requested storage area.

Return Codes

Code	Description
WHLLOK	The requested storage was allocated.

WHLLBLOCKNOTAVAIL	You requested more storage than is available.
WHLLSYSERROR	The function failed due to a system error. Any time results are unpredictable.
WHLLNOTSUPPORTED	The function was not supported by the emulation program.

Free Storage

The Free Storage sub-function frees the block of storage allocated by the Get Storage sub-function.

Function Call **WinHLLAPI(STORAGEMGR,*lpbyString*,*lpwLength*,FREESTORAGE)**

Call Parameters

Parameter	Description
Data String	NA
Data Length	Storage Block ID of area to be freed.
PS Position	02 (**FREESTORAGE**)

Return Parameters

Parameter	Description
WHLLOK	The specified block was freed.
WHLLNOTCONNECTED	You requested more storage than is available.
WHLLSYSERROR	The function failed due to a system error. Any time results are unpredictable.
WHLLNOTSUPPORTED	The function was not supported by the emulation program.

Free All Storage

The Free All Storage sub-function frees all allocated blocks of storage.

Function Call **WinHLLAPI(STORAGEMGR,*lpbyString*,*lpwLength*,FREEALLSTORAGE)**

Call Parameters

Parameter	Description
Data String	NA
Data Length	NA
PS Position	04 (**FREEALLSTORAGE**)

Return Parameters

Parameter	Description
WHLLOK	The function was successful. All blocks have been freed.
WHLLSYSERROR	The function failed due to a system error. Any time results are unpredictable.
WHLLNOTSUPPORTED	The function was not supported by the emulation program.

Query Free Storage

The Query Free Storage sub-function returns the size (in bytes) of the largest single available block of storage available to the calling Windows HLLAPI application. This value must be over 16 to be used by the Get Storage sub-function.

Function Call **WinHLLAPI(STORAGEMGR,*lpbyString*,*lpwLength*,**
QUERYFREESTORAGE)

Call Parameters

Parameter	Description
Data String	NA
Data Length	NA
PS Position	03 (**QUERYFREESTORAGE**)

Return Parameters

Parameter	Description
Data Length	Size of the largest block available. (0xFFFF indicates a full 64 Kbytes)

Return Codes

Code	Description
WHLLOK	The Query was successful.
WHLLSYSERROR	The function failed due to a system error. Any time results are unpredictable.
WHLLNOTSUPPORTED	The function was not supported by the emulation program.

Write Structured Fields—Function 127

This function writes structured field data from the Windows HLLAPI application to the host application.
If the call specifies asynchronous (A), the application receives control as soon as the request has been successfully queued to the subsystem. If the call specifies synchronous (S), WinHLLAPI waits for the host to acknowledge receipt of data before returning control to the application
The application provides the buffer address from which data is sent to the host. The buffer must be obtained using the Allocate Communications Buffer (123) function call.

Prerequisite Functions

Connect Structured Fields (function 120).
Allocate Communications Buffer (function 123).

Function Call

WinHLLAPI(WRITESTRFLDS,*lpbyString***,***lpwLength***,***lpwReturnnCode***)**

Call Parameters

Parameter	Description	
Data String	An 8-byte string for synchronous, or a 14-byte string for asynchronous in the following format:	
	Byte 1	A 1-character session short name.
	Bytes 2	A 1-character specifying the control option: u S (synchronous control) - control is not returned to the application until the read is satisfied. u A (asynchronous control) - control is returned immediately to the application.
	Bytes 3-4	The 16-bit word unique destination/origin ID returned by the Connect Structured Fields (120) function call.
	Bytes 5-8	The 4-byte value of the buffer address into which the data is to be read. The buffer must be obtained using the Allocate Communications Buffer (123) function call.
Data Length	Must be 8 or 14.	
PS Position	NA.	

Return Parameters

Parameter	Description
Data String	When the A (asynchronous) control option is specified and the request is successfully completed, the following are returned:

Bytes 9-10	A 16-bit value representing the destination/origin ID returned to the application by the emulator. This function request ID is used by the Get Request Completion (125) function to determine the status of this function call.
Bytes 11-14	A 4-byte value in which the semaphore address is returned by WINHLLAPI. The application may wait upon this semaphore. When the semaphore is cleared, the application must issue the Get Request Completion (125) function call.

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

Note There is no returned data string for the S (synchronous) control option.

Return Codes

Code	Description
WHLLOK	The function was successful.
WHLLNOTCONNECTED	An invalid presentation space was specified, or was not connected or the DOID was incorrect.
WHLLPARAMETERERROR	An invalid option was specified.
WHLLSYSERROR	The function failed due to a system error.
WHLLUNAVAILABLE	The requested resource was not available.
WHLLTRANCANCLI	An inbound transmission to the host was canceled.
WHLLTRANCANCL	An outbound transmission from the host was canceled.
WHLLHOSTCLOST	Contact with the host was lost.
WHLLOKDISABLED	The function was successful. ***Warning:*** The host inbound transmission is disabled.

The application must correct the situation if one of the following return codes is specified:

Code	Description
WHLLTRANCANCL	Is returned if the first Read Structured Fields (126) or Write Structured Fields (127) is requested after an outbound transmission from the host is canceled.
WHLLHOSTCLOST	Which requires the application to disconnect from the communications subsystem and reconnect to establish communications with the host again.
WHLLOKDISABLED	Which is returned if the host is inbound disabled.

Remarks

When the call to Read Structured Fields (126) is complete, the Read Buffer, whose address was specified in positions 5-8 of the data string, will contain the structured fields received from the host application.

The format of the Read Buffer is as follows:

Position	Meaning
Bytes 0-1	A 16-bit value - 0x0000.
Bytes 2-3	A 16-bit value which contains the Message length (*m*), which is the number of bytes of data in the message not including the 8-byte message header. This value is returned by the subsystem.
Bytes 4-5	A 16-bit value which contains the Buffer Size (*n*), which is the supplied length of the data buffer not including the 8-byte message header.
Bytes 6-7	A 16-bit value - 0xC000
Bytes 8-9	A 16-bit value which contains the Length of the first or only structured field message (not byte-reversed).
Byte 10	First non-length byte of the structured field message.
Byte *m*	Last byte in the structured field message.

Bytes 0-7 are the buffer header, which is passed to and used by the communications subsystem. The application must prepare the buffer header before using it in the structured fields call. The word at position 0 must be set to a value of zero. The length of the buffer, requested with the Allocate Communications Buffer (123) function, must be in the word at position 4. The word at position 6 must be set to 0xC000.

Bytes 8-*m* are where the structured field messages are returned. The following occurs when the call is returned:

- The word at position 2 contains the length (8-*m*) of the structured field messages.
- The word at position 8 contains the length of the first structured field message.
- Bytes 10-*m* contain the actual data of the structured field message.

Synchronous Requests

When Write Structured Fields (127) is requested synchronously (the S option in the data string), control is returned to the application only after the request is satisfied. The application can assume:

- The return code is correct.
- The data in the communications buffer (read buffer) is correct.
- The host is no longer processing the Write Structured Fields (127) request.

Asynchronous Requests

When Write Structured Fields (127) is requested asynchronously (the A option in the data string), the application cannot assume:

- The return code is correct.
- The data in the communications buffer (read buffer) is correct.
- The host is no longer processing the Write Structured Fields (127) request.

When requested asynchronously, WinHLLAPI returns the following:

- A 16-bit Request ID in positions 2-3 of the data string.
- The address of a semaphore in positions 4 - 7 of the data string.

These are used to complete the asynchronous Write Structured Fields (127) call.

The following steps must be completed to determine the outcome of an asynchronous Write Structured Fields (127) function call:

- If the WinHLLAPI return code is not zero, the request failed. No asynchronous request has been made. The application must take appropriate actions before attempting the call again.
- If the return code is zero, the application should wait until the semaphore is cleared by using the Get Request Completion (125) function. The semaphore should not be freed (this done by the Get Request Completion (125) function) and should not be reused. The semaphore is only valid for the duration of the Read Structured Fields (126) function call through the completion of the Get Request Completion (125) function call.
- Once the semaphore is cleared, use the returned 16-bit Request ID as the Request ID parameter in a call to the Get Request Completion (125) function. The data string returned from the Get Request Completion (125) function call contains the final return code of the Write Structured Fields (127) function call.

Note The communications subsystem allows for a maximum of 20 asynchronous requests per application to be outstanding. A return code for unavailable resources (RC= 11) is returned if more than 20 asynchronous requests are attempted.

Note Structured fields are not supported by the COBOL programming language due to memory access problems inherent to the language.

APPENDIX D

Query Reply Data Structures for Windows HLLAPI

This appendix lists and defines the query reply structures supported by the Windows HLLAPI structured field interface. See the IBM 3270 Information Display System Data Stream Programmer's Guide for additional information on Query Reply Data Structures.

1. WinHLLAPI must scan the query reply buffers to locate the destination/origin ID (DOID) self-defining parameter (SDP) for the structured field support to work and be reliable. The DOID field is then filled in with the assigned ID.
2. The application should build the query reply data structures in the application's private memory.
3. Only cursory checking is performed on the query reply data. Only the ID and the length of the structure are checked for validity.
4. The 2-byte length field at the beginning of each query reply is not byte-reversed.
5. Only one distributed data management (DDM) base-type connection is allowed per host session. If the DDM connection supports the SDP for the DOID, then multiple connections are allowed.
6. If a nonzero return code is received indicating that an application is already connected to the selected session (RC 32 or 39), use of that presentation space should be with caution. Conflicts file transfer and other Windows HLLAPI applications may result.

The DDM Query Reply

Several DDM query reply formats will be supported. Some of these formats are listed below:

Table D-l. DDM Query Reply Base Format

Offset	Length	Content	Meaning
0	1 word	Length	Length of Structure
2	1 byte	0x81	Query Reply ID
3	1 byte	0x95	Query Reply Type
4-5	2 bytes	FLAGS	Reserved
6-7	2 bytes	LIMIN	Maximum DDM bytes allowed in inbound transmission
8-9	2 bytes	LIMOUT	Maximum DDM bytes allowed in outbound transmission
10	1 byte	NSS	Number of subsets identifier
11	1 byte	DDMSS	DDM subset identifier

DDM Application Name Self-Defining Parameter

The DDM Application Name self-defining parameter provides the host application with the name of the application containing control of the DDM Auxiliary Device. The controlling application is identified by the DOID in the Direct Access self-defining parameter.

This SDP is optional, but it is necessary if a host application is to identify distinct DDM auxiliary devices when more than one application is in existence at a remote workstation.

Table D-2. DDM Application Name SDP

Offset	Length	Content	Meaning
0	1 byte	Length	Parameter Length
1	1 byte	0x02	DDM Application Name
2-n	n-2 bytes	NAME	Name of Remote Application Program

NAME: The name consists of 8 characters or less and is the means by which a host application may relate to an application in a remote workstation. It is the responsibility of the host and remote application users to ensure that the name is understood by the applications at each end.

PCLK Protocol Controls Self-Defining Parameter

The PCLK (PC link) Protocol Controls self-defining parameter indicates that the PCLK Protocol Controls structured field, ID = X'1013', can be used both inbound and outbound in data streams destined to or from the DDM auxiliary device processor.

Table D-3. DDM PCLK Auxiliary Device SDP

Offset	Length	Content	Meaning
0	1 byte	0x04	Parameter Length
1	1 byte	0x03	PCLK Protocol Controls
2-n	n-2 bytes	VERS	Protocol Version

VERS: The value given in VERS is used to indicate the version of PCLK installed in the terminal at the time the query reply is returned. For example, 0x0001 indicates PCLK version 1.1.
See the *IBM 3270 Information Display System Data Stream Programmer's Reference* for the field definitions for this query reply.

Base DDM Query Reply Formats

The following query reply formats are examples of some of the Base + SDP combinations possible. Not all of the combinations are shown.

Table D-4. Base DDM Query Reply Format with Name and Direct Access SDPs

Offset	Length	Content	Meaning
0	1 word	Length	Length of structure (includes SDPs)
2	1 byte	0x81	Query Reply ID
3	1 byte	0x95	Query Reply Type
4-5	2 bytes	FLAGS	Reserved
6-7	2 bytes	LIMIN	Maximum DDM bytes allowed in inbound transmission
8-9	2 bytes	LIMOUT	Maximum DDM bytes allowed in outbound transmission
10	1 byte	NSS	Number of subsets supported
11	1 byte	DDMSS	DDM subset identifier
12	1 byte	Length (n+2)	Parameter Length
13	1 byte	0x02	DDM Application Name

Table D-4. Base DDM Query Reply Format with Name and Direct Access SDPs (*continued*)

Offset	Length	Content	Meaning
14 -(13+n)	n bytes	Name	Name of the Remote Application Program
14 + n	1 byte	0x04	Parameter Length
15 + n	1 byte	0x01	Direct Access ID
16+n - 17+n	n-2 bytes	VERS	Destination/Origin ID assigned by the subsystem

The SDPs begin at offsets 12 and (14 + n) where "n" is the length of the application name supplied at offset 14.
See the *IBM 3270 Information Display System Data Stream Programmer's Reference* for the field definitions for this query reply.

Table D-5. Base DDM Query Reply Format with Direct Access and Name SDPs

Offset	Length	Content	Meaning
0	1 word	Length	Length of structure (includes SDPs)
2	1 byte	0x81	Query Reply ID
3	1 byte	0x95	Query Reply Type
4-5	2 bytes	FLAGS	Reserved
6-7	2 bytes	LIMIN	Maximum DDM bytes allowed in inbound transmission
8-9	2 bytes	LIMOUT	Maximum DDM bytes allowed in outbound transmission
10	1 byte	NSS	Number of subsets supported
11	1 byte	DDMSS	DDM subset identifier
12	1 byte	0x04	Parameter Length
13	1 byte	0x01	Direct Access ID
14-15	2 bytes	DOID	Destination/Origin ID assigned by the subsystem
16	1 byte	Length (n+2)	Parameter Length
17	1 byte	0x02	Direct Access ID
16+n - 17+n	n bytes	Name	Name of the Remote Application Program

The SDPs begin at offsets 12 and 16.
See the *IBM 3270 Information Display System Data Stream Programmer's Reference* for the field definitions for this query reply.

The IBM Auxiliary Device Query Reply

The Auxiliary Device Query Reply is used to indicate to the host application the support of an IBM auxiliary device, which uses a data stream defined by IBM. See the *IBM 3270 Data Stream Programmer's Reference Manual* for more information.
When the function is supported, the query reply is transmitted inbound in reply to a Read Partition structured field specifying Query or Query List (QCODE List = 0x9E, Equivalent, or All).
When a workstation supports multiple auxiliary devices, the IBM Auxiliary Device Query Reply must be sent for each of the devices.
Optional Parameters: All parameters shown in the base part of the query reply must be present. Parameters not used are set to 0x100. At least one self-defining parameter must be present.

Table D-6. IBM Auxiliary Device Base Format with Direct Access SDP

Offset	Length	Content	Meaning
0	1 word	Length	Length of structure (includes SDPs)
2	1 byte	0x81	Query Reply ID
3	1 byte	0x9E	IBM Auxiliary Device Query Reply
4	1 byte	Bit 0	
Bits 1-7	FLAGS	QUERY Binary 1	
RES	Reserved	Read Part (Query, Query List)	IBM Auxiliary device supports Query
Reserved, must be binary 0's	5	1 byte	FLAGS
Reserved	6-7	2 bytes	LIMIN
Maximum DDM bytes allowed in inbound transmission	8-9	2 bytes	LIMOUT
Maximum DDM bytes allowed in outbound transmission	10	1 byte	TYPE

Table D-6. IBM Auxiliary Device Base Format with Direct Access SDP (*continued*)

Offset	Length	Content	Meaning
0x01	0x02	Others	Type of auxiliary device supported
IBM auxiliary device display	IBM auxiliary device printer	Reserved	11
1 byte	0x04	Parameter Length	12

1 byte	0x01	Direct Access	13-14
1 word	DOID	Destination/Origin ID assigned by the subsystem	

QUERY
: This bit must be set to Binary 1 for all IBM auxiliary devices to indicate that it supports receiving a Read Partition (Query, Query List). The host application may then use a Read Partition directed to the auxiliary device to determine its characteristics. The destination/origin structured field is used to direct the Read Partition structured field to the auxiliary device.

  The minimum support level for the IBM auxiliary device is to return the Null Query Reply in response to the Read Partition.

LIMIN
: States the maximum number of bytes that can be sent in an inbound transmission. A LIMIN value of X'0000' indicates no implementation limit on the number of bytes transmitted inbound.

LIMOUT
: States the maximum number of bytes that can be sent to the IBM auxiliary device in an outbound transmission. A LIMOUT value of 0x0000 indicates no implementation limit on the number of bytes transmitted outbound.

TYPE
: Identifies the auxiliary device being supported. Two values are valid. One identifies an auxiliary display and the other identifies an auxiliary printer. All other values are reserved.

  The IBM auxiliary device processor supports two Self-Defining Parameters, 01 and 03. These are defined in Table D-7.

Direct Access Self-Defining Parameter

This self-defining parameter provides the ID for use in the destination/origin structured field in the direct access of the IBM auxiliary device.
This SDP is always required to accompany the base query reply.

Table D-7. IBM Auxiliary Device Direct Access SDP

Offset	Length	Content	Meaning
0	1 byte	0x04	Parameter Length
1	1 byte	0x03	PCLK Protocol Controls
2-3	2 bytes	DOID	Destination/Origin ID

DOID: The value in these bytes is used in the ID field of the destination/origin structured field to identify the auxiliary device as the destination or origin of the data which follows.

PCLK Protocol Controls Self-Defining Parameter

The presence of the PCLK Protocol Controls self-defining parameter indicates that the PCLK Protocol Controls structured field, ID = 0x1013, can be used both inbound and outbound in data streams destined to or from the IBM auxiliary device processor.

Table D-8. IBM Auxiliary Device PCLK SDP

Offset	Length	Content	Meaning
0	1 byte	0x04	Parameter Length
1	1 byte	0x03	PCLK Protocol Controls
2-3	2 bytes	VERS	Protocol Version

VERS: The value given in VERS is used to indicate the version of PCLK installed in the terminal at the time the query reply is returned. For example, 0x000l indicates PCLK version 1.1.
See the *IBM 3270 Information Display System Data Stream Programmer's Reference* for the field definitions for this query reply.

The OEM Auxiliary Device Query Reply

The OEM Auxiliary Device Query Reply format is as follows:

Table D-9. OEM Auxiliary Device Base Format with Direct Access SDP

Offset	Length	Content	Meaning
0	1 word	0x001A	Length of structure (includes SDPs)
2	1 byte	0x81	Query Reply ID
3	1 byte	0x8F	OEM Query Reply
4-5	2 bytes	FLAGS	Reserved
6-13	4 words	DTYPE	Device Type
14-21	4 words	UNAME	User assigned name
22	1 byte	0x04	Parameter Length
23	1 byte	0x01	Direct Access
24-25	1 word	DOID	Destination/Origin ID assigned by the subsystem

See the *IBM 3270 Information Display System Data Stream Programmer's Reference* for the field definitions for this query reply.

The OEM auxiliary device processor supports two Self-Defining Parameters, 01 and 03. These are defined in Table D-10.

Direct Access Self-Defining Parameter

This self-defining parameter provides the ID for use in the destination/origin structured field in the direct access of the OEM auxiliary device.

This SDP is always required to accompany the base query reply.

Table D-10. OEM Auxiliary Device Direct Access SDP

Offset	Length	Content	Meaning
0	1 byte	0x04	Parameter Length
1	1 byte	0x01	Direct Access ID
2-3	2 bytes	DOID	Destination/Origin ID

DOID: The value in these bytes is used in the ID field of the destination/origin structured field to identify the auxiliary device as the destination or origin of the data which follows.

PCLK Protocol Controls Self-Defining Parameter

The presence of the PCLK Protocol Controls self-defining parameter indicates that the PCLK Protocol Controls structured field, ID = 0x1013, can be used both inbound and outbound in data streams destined to or from the IBM auxiliary device processor.

Table D-ll. OEM Auxiliary Device PCLK SDP

Offset	Length	Content	Meaning
0	1 byte	0x04	Parameter Length
1	1 byte	0x03	PCLK Protocol Controls

2-3	2 bytes	VERS	Protocol Version

VERS: The value given in VERS is used to indicate the version of PCLK installed in the terminal at the time the query reply is returned. For example, 0x000l indicates PCLK version 1.1.

The Cooperative Processing Requester Query Reply

The Cooperative Processing Requester query reply is also called the SRPI query reply or the CPSI query reply. The format is as follows:

Table D-12. CPR Query Reply Buffer Format

Offset	Length	Content	Meaning
0	1 word	Length	Length of structure (includes SDPs)
2	1 byte	0x81	Query Reply ID
3	1 byte	0xAB	Query Reply Type
4-5	2 bytes	FLAGS	Reserved
6-7	1 word	LIMIN	Maximum number bytes allowed in inbound transmission
8-9	1 word	LIMOUT	Maximum number bytes allowed in outbound transmission
10	1 byte	FEATL	Length in bytes of the following feature information
11-12	1 word	FEATS	CPR Length and feature flags
13 to (N*2)+12	0-2 bytes	FEATSs	Additional flags
(N*2)+12	1 byte	0x04	Length of DOID SDP
(N*2)+13	1 byte	0x01	Type for Destination/Origin ID
(N*2)+14	1 word	DOID	Destination/Origin ID assigned by the subsystem

See the *IBM 3270 Information Display System Data Stream Programmer's Reference* for the field definitions for this query reply.

The Product Defined Query Reply

This query reply is used by IBM products using registered subidentifiers within the X ' 9C ' data structure. The Product Defined Data Stream query reply indicates support of a 3270DS workstation auxiliary device which uses an IBM product defined data stream. The data stream is not defined by a format architecture document having an identifiable control point such as an architecture review board. When an auxiliary device supports an IBM product defined data stream, this query reply is transmitted inbound in reply to a Query List (QCODE List = 0x9C or All).

Optional Parameters: All parameters shown in the base part of the query reply and the Direct Access self-defining parameter must be present.

The format of the Product Defined query reply is as follows:

Table D-13. IBM Product Defined Query Reply Base Format

Offset	Length	Content	Meaning
0	1 word	Length	Length of structure (includes SDPs)
2	1 byte	0x81	Query Reply ID
3	1 byte	0x9C	IBM Product Defined Data Stream
4-5	2 bytes	FLAGS	Reserved
6	1 byte	REFID	Reference Identifier
7	1 byte	SSID	Subset Identifier
8	1 byte	0x04	Parameter Length
9	1 byte	0x01	Direct Access
10-11	1 word	DOID	Destination/Origin ID assigned by the subsystem

Valid values for REFID (offset 6) and SSID (offset 7) of the Product Defined query reply are as follows:

Table D-14. IBM Product Defined Query Reply Base Format

REFID	SSID	Product and Data Stream Documentation
0x01		5080 Graphics System: This reference ID indicates the 5080 Graphics System data stream is supported by the auxiliary device. Descriptions of the 5080 Graphics Architecture, structured fields, subset IDs, DOID and associated function sets, are defined in the following: IBM 5080 Graphics System Principles of Operation
	0x001	5080 HGFD Graphics Subset
	0x002	5080 RS232 Ports Subset
0x02		WHIP API (replaced by SRL name when written) This reference ID indicates that the WHIP API data stream is supported by the auxiliary device. A description of the WHIP API architecture is defined in the following: IBM RT PC Workstation Host Interface Program Version 1.1 User's Guide and Reference Manual
	0x001	WHIP Subset 1
0x03 to 0xFF		All other values are reserved

The IBM Product Defined processor supports only the Direct Access Self-Defining Parameter. It is defined in Table D-15.

Direct Access Self-Defining Parameter

The presence of the Direct Access ID self defining parameter indicates the auxiliary device may be accessed directly by using the destination/origin structured field. When multiple auxiliary devices are supported which use a product defined data stream, separate Product Defined Data Stream query replies must be provided, each of which has a unique DOID.

Table D-15. IBM Product Defined Direct Access SDP

Offset	Length	Content	Meaning
0	1 byte	0x04	Parameter Length
1	1 byte	0x03	PCLK Protocol Controls
2-3	2 bytes	DOID	Destination/Origin ID

DOID: The value in these bytes is used in the ID field of the destination/origin structured field to identify the auxiliary device as the destination or origin of the data which follows.

The Document Interchange Architecture Query Reply

This query reply indicates the Document Interchange Architecture (DIA) function sets supported. The format of the DIA Query Reply is as follows:

Table D-16. IBM DIA Base Format

Offset	Length	Content	Meaning
0	1 word	Length	Length of structure (includes SDPs)
2	1 byte	0x81	Query Reply ID
3	1 byte	0x97	IBM Product Defined Data Stream
4-5	2 bytes	FLAGS	Reserved
6-7	2 bytes	LIMIN	Maximum message bytes allowed in inbound transmission
8-9	2 bytes	LIMOUT	Maximum message bytes allowed in outbound transmission
10	1 byte	NFS	Number of 3-byte function set ID's which follow
11-13	3 bytes	DIAFS	DIA function set identifier
14 - (13+(N*3))	N*3 bytes	DIAFSs	Additional DIA function set ID's
14+(N*3)	1 byte	0x04	Parameter Length
15+(N*3)	1 byte	0x01	Direct Access
16+(N*3)	1 word	DOID	Destination/Origin ID assigned by the subsystem

The DIA auxiliary device processor supports only the Direct Access Self-Defining Parameter. It is defined in Table D-17.

Direct Access Self-Defining Parameter

The presence of the Direct Access ID self defining parameter indicates the auxiliary device may be accessed directly by using the destination/origin structured field.

Table D-17. DIA Auxiliary Device Direct Access SDP

Offset	Length	Content	Meaning
0	1 byte	0x04	Parameter Length
1	1 byte	0x01	Direct Access ID
2-3	2 bytes	DOID	Destination/Origin ID

DOID: The value in these bytes is used in the ID field of the destination/origin structured field to identify the auxiliary device as the destination or origin of the data which follows.
See the *IBM 3270 Information Display System Data Stream Programmer's Reference* for the field definitions for this query reply.