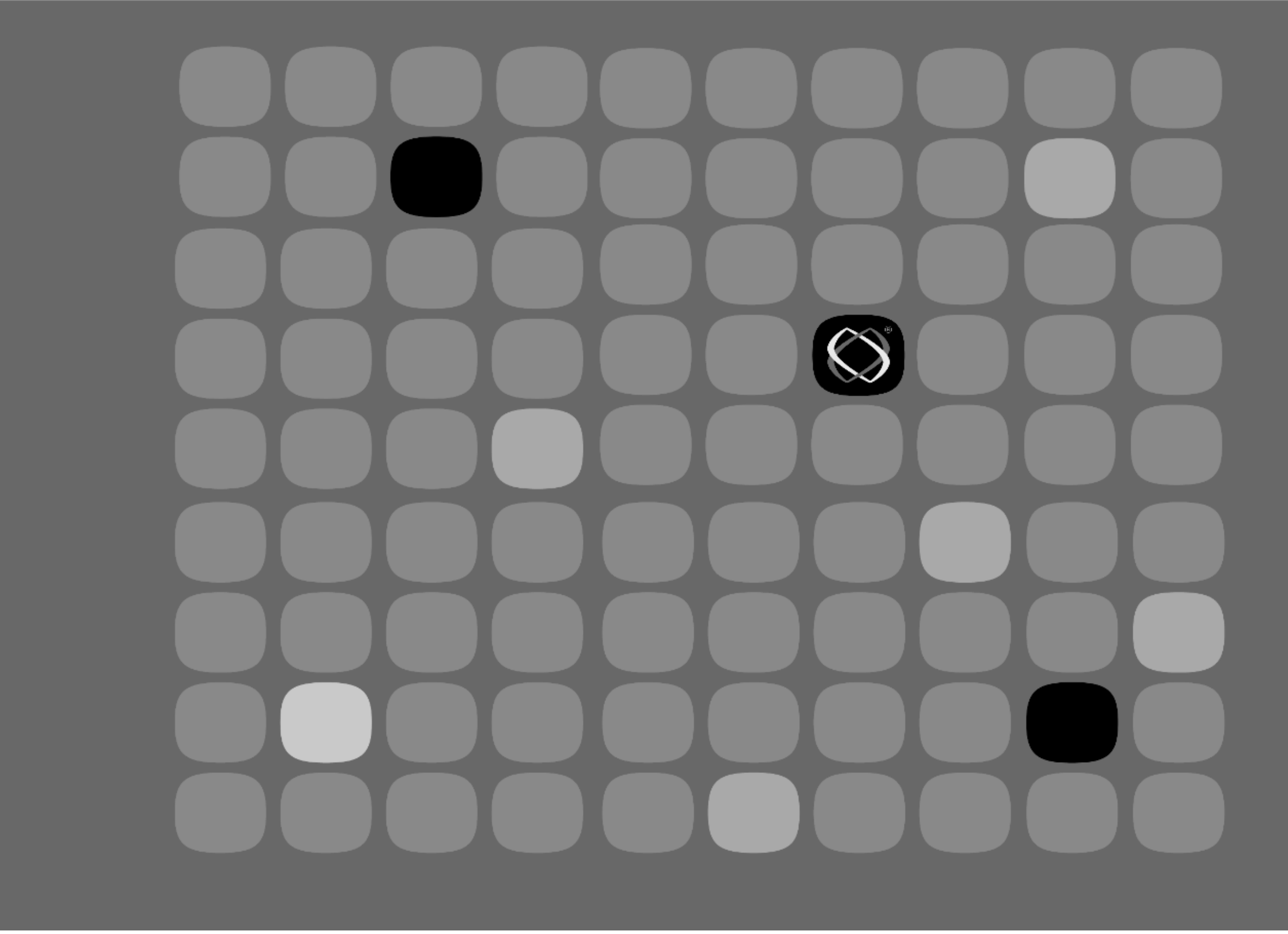


Programmer's Reference

Attachmate®

INFOCONNECT®

Enterprise Edition

IHLLAPI

Contents

About This Guide

This guide describes the INFOConnect™ High-Level Language Application Program Interface (IHLLAPI). It covers programming guidelines and IHLLAPI functions, and is intended for programmers writing IHLLAPI applications.

Conventions

This guide uses the following documentation conventions:

- Text that you type as well as messages and prompts that appear on the screen appear `in this type style`.
- In addition to emphasizing text and highlighting terms used for the first time, *italics* indicate variables. For example, if you were asked to type *drive*:\directory\filename.ext, you would enter the actual drive, directory, and file name in place of the italicized words.
- The word *PC* refers to any personal computer running Windows® 7, Windows Vista, Windows XP or Windows 2000.
- The word *host* refers to any mainframe, mini-computer, or information hub with which the PC communicates.

Abbreviations

The following is an alphabetical list of common abbreviations and acronyms used in this guide. Please note that abbreviations and acronyms are not generally spelled out in the text. Refer back to this table as necessary.

Abbreviation/ Acronym	Meaning
ADK	Automation Development Kit
AID	Attention identification
DLL	Dynamic link library
EAB	Extended attribute byte
EHLLAPI	Emulator high-level language application programming interface
EOT	End-of-text
HLLAPI	High-level language application programming interface
IHLLAPI	INFOConnect high-level language application programming interface
LTAI	Line transmission activity indicator
MDI	Multiple document interface
MDT	Modified data tag
OIA	Operator information area (status line)
PS	Presentation space

Related Documentation

Additional information exists in the form of Readme files and guides.

Readme Files

README.ADK contains important notices, known limitations, and the latest information about IHLLAPI that could not be included in this guide.

If you double-click this file in My Computer or File Manager, your PC might not recognize the file extension. Select Notepad as the application to use to read the file.

Guides

For more information, refer to the following documentation:

- The *Getting Started* guide explains how to install your INFOConnect products and get them up and running.
- The *INFOConnect Connectivity Services Installation, Configuration, and Operations Guide* provides information about INFOConnect paths, path templates, libraries, and accessories.

Introduction 1

In This Chapter

This chapter provides an overview of INFOConnect HLLAPI as well as the concepts and terms you'll need to understand as you work with the functions. It includes the following headings:

What Is IHLLAPI?

INFOConnect HLLAPI (IHLLAPI) is a set of function calls that you can use to write Windows applications that interact with Attachmate's terminal emulators (such as T 27 and UTS), and therefore with host applications.

The primary IHLLAPI module is a dynamic link library (IHLAPI32.DLL) that is automatically or explicitly linked to your application. This file and other IHLLAPI files are installed in the ACCMGR32 folder within the INFOConnect folder when you install any of Attachmate's emulators (which are purchased separately). You can use the IHLAPI32.DLL when writing applications using Visual Basic®, C, or any language that provides interfaces for Windows DLLs.

The Automation Development Kit (ADK) also includes the following additional files, which are installed in the ADK32 folder of your INFOConnect folder:

File	Description
HLLAPI.H	A header file that must be included with applications written in C, C++, or any other language that can include a header file For information about this file, refer to "HLLAPI.H Header File" on page 173.
HLLAPI32.LIB	The import library included in C and C++ applications in their link list to supply the names and locations of functions that are exported by the HLLAPI32.DLL
HLLWATCH.EXE	An application run by the IHLAPI32.DLL that intercepts messages from Accessory Manager for host notification and keystroke intercept functions
IHLAPI32.H	A header file that must be included with applications written in C, C++, or any other language that can include a header file For information about this file, refer to "IHLAPI32.H Header File" on page 187.
IHLAPI32.LIB	The import library included in C and C++ applications in their link list to supply the names and locations of functions that are exported by the IHLAPI32.DLL

File	Description
IHLDEM32.EXE	An application that you can use to test IHLLAPI calls For information about this application, refer to "Trying Out IHLLAPI Functions" on page 15
IHLLAPI.LIB	The import library included in C and C++ applications in their link list to supply the names and locations of functions that are exported by the IHLLAPI.DLL.
VBHLLAPI.BAS	A base file that lists the IHLLAPI functions You can include this file with a Visual Basic project.
WHLLAPI.H	A header file that must be included with applications written in C, C++, or any other language that can include a header file For information about this file, refer to "WHLLAPI.H Header File" on page 168.

Differences from Other Versions of HLLAPI

IHLLAPI includes most of the functions from Microsoft's WinHLLAPI and IBM's EHLLAPI, and most IHLLAPI calls work exactly as described in the *Microsoft WinHLLAPI Specification*. However, there are a several differences between IHLLAPI and other HLLAPI implementations.

For example, several IHLLAPI functions use the standard WinHLLAPI return codes differently, or use other return codes in addition to the standard codes. (Refer to Appendix A, "Return Codes," for more information about the return codes.)

In addition, since Unisys® terminal emulators have different characteristics than IBM terminal emulators, IHLLAPI differs from EHLLAPI in how it handles field attributes and status line information.

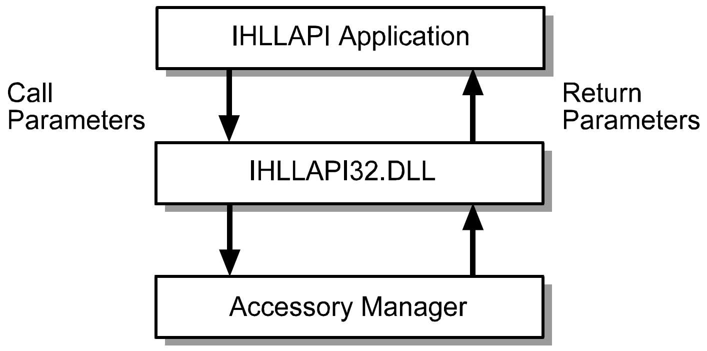
IHLLAPI also includes additional functions (such as Associate Profile) that support the IHLAPI32.DLL's interactions with Accessory Manager. These functions are not defined within the Microsoft WinHLLAPI specification but are described in this guide.

How IHLLAPI Works

When your application calls an IHLLAPI function, the following actions occur:

- If the IHLAPI32.DLL is not already in memory, Windows loads it.
- The application passes the call parameters to the IHLAPI32.DLL.
- The IHLAPI32.DLL interprets the request.
- The IHLAPI32.DLL sends the request to a terminal emulation session (a T 27 session or UTS session) running within Accessory Manager.
- The emulator performs any necessary host communication, processes the request, and returns the result of the request to the IHLAPI32.DLL.
- The IHLAPI32.DLL sends the results back to the application through the return parameters.

The following figure shows the flow of IHLLAPI functions:



When your application closes, Windows removes the IHLAPI32.DLL from memory.

About Short Names and Session File Names

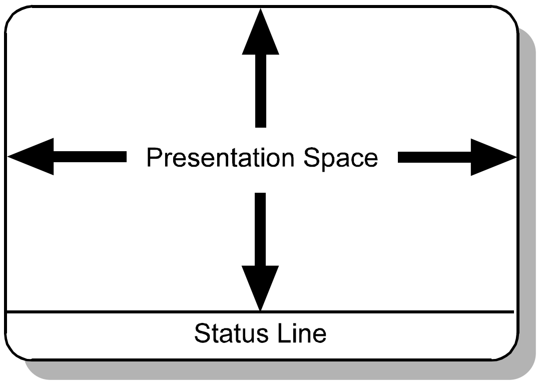
IHLLAPI functions refer to sessions by short names (the letters A through Z) rather than file names (such as SESSION1.ADP).

To use IHLLAPI applications with a T 27 or UTS session, you must associate a short name with an existing session file name. For example, you can associate the short name A with the session file name SESSION1.ADP. Whenever an IHLLAPI function uses the short name A, that function is performed using SESSION1.ADP.

You can associate short names with session file names using Accessory Manager or using an IHLLAPI function call from your application. The procedure for doing this using Accessory Manager is provided in "Using Short Names" on page 20. For information about doing this using an IHLLAPI function call, refer to "Associate Profile (911)" on page 147.

About the Presentation Space

Each session that your IHLLAPI application can connect with is configured to use a specific number of columns and rows that define the area within which the host or user can display or type data. For IHLLAPI functions, this area is called the *presentation space*:



Many IHLLAPI functions interact with the presentation space. For example, your IHLLAPI application can perform the following tasks:

- Search the presentation space for certain strings to determine which host application screen is displayed
- Position the cursor within the presentation space and send keystrokes to navigate among host application screens or retrieve data
- Copy data from the presentation space and display them in your IHLLAPI application

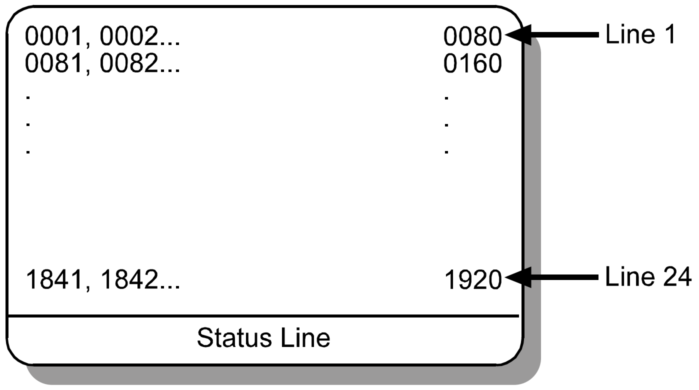
The status line that appears at the bottom of the screen (also known as the operator information area, or OIA) is not included in the presentation space. There are separate functions for accessing the presentation space and the OIA.

Presentation Space Positions

When specifying a location within a presentation space, IHLLAPI functions use presentation space positions rather than column and row coordinates.

Presentation space positions start at position 1 and end with the maximum number of presentation space positions for the emulator. For example, a session with 24 rows and 80 columns has 1920 presentation space positions (24x80); a session with 24 rows and 132 columns has 3168 presentation space positions (24x132).

The following figure shows the presentation space positions for an emulator configured for 24 rows and 80 columns:



For sessions that have multiple pages, each page is treated like a new presentation space. For example, in a session configured for 24 rows, 80 columns, and two pages, the first page would contain presentation space positions 1 through 1920, and the second page would also contain positions 1 through 1920.

Standard Presentation Space Sizes

Different emulators support different presentation space sizes. For example, T 27 supports up to 255 rows and 132 columns; UTS supports up to 255 rows and 256 columns. Although you can configure the size of the presentation space, the following table shows the standard sizes that are generally associated with each emulator:

Emulator	Rows	Columns	Total Size
T 27	24	80	1920
UTS	24	80	1920
	24	132	3168

The row totals shown in the preceding table do not include the status line. The status line appears at the bottom of each terminal emulation screen and contains information about the session status (such as what mode the emulator is in) and other information (such as the location of the cursor). T 27 can also display a line of information between the presentation space and the status line; this is known as the user message.

Calculating the Presentation Space Position

You can use the Convert Position or RowCol (99) function to convert a presentation space position into row and column coordinates. For example, for a session with 80 columns, presentation space position 160 would occur at row 2, column 80. However, for a session with 132 columns, presentation space 160 would occur at row 2, column 28.

You can also use this function to converts row and column coordinates into a presentation space position. In the preceding example, this function could convert row 2, column 80, to presentation space position 160.

For more information about this function, refer to "Send File (90)" on page 128.

About Fields and Attributes

A presentation space can be either unformatted or formatted. In an unformatted presentation space, the data generally appear as continuous lines of text on the screen. In a formatted presentation space, the data generally appear in individual fields.

Each field has attributes that define the type of information that can be typed in that field, as well as the appearance of the field on the screen. For example, field attributes can specify the following types of information:

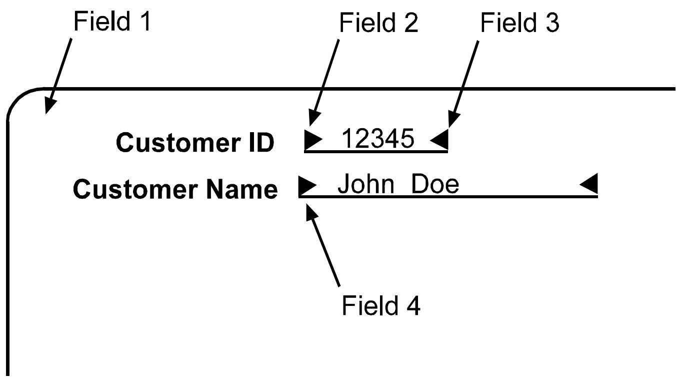
- Whether the field is protected (read-only) or unprotected (read and write)
- Type of data allowed in the field (alphanumeric or numeric)
- Appearance of the field (color, blinking, reverse video)
- Whether the field has been modified since it was put on the presentation space

Many IHLLAPI functions interact with fields. For example, your IHLLAPI application can perform the following tasks:

- Determine the attributes of a particular field (such as whether it is protected or unprotected)
- Copy data from your application to a particular field
- Copy data from a particular field to your application

Identifying Fields

The IHLAPI32.DLL identifies fields by detecting certain changes in the attributes of each field. For example, the following figure shows how fields would be identified in an T 27 session:



In this example, the first field begins at the upper left corner of the screen and contains empty spaces and the label Customer ID. The second field includes all the data between the delimiters. The third field begins at the end of the delimiter for Field 2 and contains empty spaces and the label Customer Name. The fourth field includes all the data between the second set of delimiters.

The attributes that define fields are unique to each emulator. For example, in the preceding figure, the label Customer ID might be blinking, but these character attributes do not indicate the beginning of a new field. The following table lists the attributes of each emulator and whether they denote a field:

Emulator	Denote New Field?	Attributes
T 27	YES	Non-transmittable, protected, unprotected, left and right justified
	NO	Intensity, blinking, reverse video, secure
UTS	YES	Protected, unprotected, numeric only, alphanumeric only, left and right justified
	NO	Color, emphasis

About Session Parameters

Many IHLLAPI functions are affected by options known as session parameters. This term does not refer to the configuration of the terminal emulation session with which your application communicates, but rather to options specified by the Set Session Parameters (9) function.

For example, when the disconnect session parameter is set to DISCONLOG, the Disconnect Presentation Space (2) function only disconnects your application from the session. When this parameter is set to DISCONPHYS, this function disconnects your application from the session, disconnects the session from the host, and closes the session.

For recommendations about how to use the Set Session Parameters (9) function, refer to "Setting Session Parameters" on page 25. For detailed information about this function, refer to "Set Session Parameters (9)" on page 71.

Overview of IHLLAPI Functions

The following tables provide a numerical list of the IHLLAPI functions and a brief description of each. For detailed information about these functions and how to use them, refer to Chapter , "IHLLAPI Functions."

Function	Description
Connect Presentation Space (1)	Connects your application to the specified presentation space
Disconnect Presentation Space (2)	Disconnects your application from the current presentation space
Send Key (3)	Places a keystroke or string of keystrokes in the current presentation space at the current cursor location
Wait (4)	Tests the status of the current presentation space to determine whether it can receive keystrokes
Copy Presentation Space (5)	Copies the entire presentation space to a string in your application
Search Presentation Space (6)	Scans the current presentation space for a specified string
Query Cursor Location (7)	Determines the location of the cursor in the current presentation space
Copy Presentation Space to String (8)	Copies the specified portion of the current presentation space to a string in your application
Set Session Parameters (9)	Changes the default session parameters
Query Sessions (10)	Determines the number of open sessions, as well as the short name, filename, session type, and presentation space size of each session
Query Sessions Full (910)	Same as Query Sessions, except that it returns long filenames and the full path of each session.
Copy OIA (13)	Copies the operator information area (status line) and other information to a string in your application
Query Field Attribute (14)	Determines the attribute of the specified field in the current presentation space
Copy String to Presentation Space (15)	Copies an ASCII string from your application to a specific location in the current presentation space

Function	Description
Pause (18)	Causes your application to wait a specific amount of time for an event to occur
Query System (20)	Determines the support level provided to your application by the underlying low-level and high-level modules (and other system-related values)
Reset System (21)	Re-initializes the default session parameters, stops host event notification, and disconnects the application from any connected sessions
Query Session Status (22)	Obtains information about a particular session, including the short name, file name, session type, session characteristics, and the number of rows and columns in the presentation space
Start Host Notification (23)	Enables notifying your IHLLAPI application of changes in the presentation space or operator information area (OIA)
Query Host Update (24)	Determines if the presentation space or OIA of the specified session has been updated since Start Host Notification (23) was called, or since the previous call of this function
Stop Host Notification (25)	Disables notifying your IHLLAPI application of changes in the presentation space or OIA
Search Field (30)	Searches the specified field within the current presentation space for a specified string
Find Field Position (31)	Determines the beginning position of the current field in the current presentation space
Find Field Length (32)	Determines the length of the current field in the current presentation space
Copy String to Field (33)	Copies characters from your application to a specified field in the current presentation space
Copy Field to String (34)	Copies all the characters from a specified field in the current presentation space to a data string in your application
Set Cursor Position (40)	Positions the cursor within the current presentation space
Start Keystroke Intercept (50)	Enables your application to intercept keystrokes typed by a user in the session window
Get Key (51)	Intercepts keystrokes from sessions that have keystroke intercept enabled and processes those keystrokes

Function	Description
Post Intercept Status (52)	Notifies the IHLAPI32.DLL that a keystroke obtained with the Get Key (51) function has been accepted or rejected
Stop Keystroke Intercept (53)	Stops your application from intercepting keystrokes
Send File (90)	Transfers a file from the local workstation to the host session.
Receive File (91)	Transfers a file from the host session to the local workstation.
Convert Position or RowCol (99)	Converts the presentation space position into row and column coordinates, or converts row and column coordinates into a presentation space position, depending on the call parameters passed by your application
Connect Window Services (101)	Connects your application to the specified presentation space
Disconnect Window Services (102)	Disconnects your application from the specified presentation space
Query Window Coordinates (103)	Requests the coordinates of the specified presentation space
Window Status (104)	Queries or changes a window's presentation space size, location, or visible state
Associate Profile (911)	Associates the specified short name with the specified session
Remove Profile (912)	Unassociates any session from the specified short name
Get Associations (913)	Retrieves a list of all short names that have been associated with sessions
Find File Name (914)	Gets the session file name for the specified short name
Find Short Name (915)	Gets the short name for the specified session file name

Trying Out IHLLAPI Functions

To see what the IHLLAPI functions do, use the test application included with the ADK. This application lets you send function calls to an emulator and see what values are returned. For example, you could see what happens when you issue a Connect Presentation Space (1) function call.

To run the test application, follow these steps:

1. Run Accessory Manager and assign a session file name to a HLLAPI short name.

   For instructions on this procedure, refer to "About Short Names and Session File Names" on page 5.

   For example, you could assign the session TCPA_1.ADP to the short name A.

2. Run IHLDEM32.EXE.

   Using My Computer or File Manager, go to the ADK32 folder within the INFOConnect folder and double-click IHLDEM32.EXE.

   > **Note:** To run this application from the ADK32 folder, the IHLAPI32.DLL must be accessible. For more information, refer to "Running Your IHLLAPI Application" on page 35. Alternatively, you can copy this application to the ACCMGR32 folder and run it from there.

3. Click the desired IHLLAPI function from the menu.

4. If the test application prompts you for any additional information, type the appropriate response for each prompt as it appears.

   The test application sends the function call to the IHLAPI32.DLL and displays any returned data.

5. To exit the test application, click Exit.

Guidelines for Developing IHLLAPI Applications

2

In This Chapter

This chapter provides general guidelines for developing IHLLAPI applications, as well specific guidelines for developing in Visual Basic and C. It includes the following headings:

Prerequisites for Using IHLLAPI

Before writing any applications using IHLLAPI, you should have a working knowledge of the following:

- Microsoft Windows 7, Vista, XP and 2000.
- An INFOConnect emulator (such as INFOConnect T 27 for Windows, INFOConnect UTS for Windows, or EXTRA!® Office for Accessory Manager)
- The programming language (Visual Basic, C, C++, or any language that provides interfaces for Windows DLLs) and the compiler or assembler you plan to use

Using Existing IHLLAPI Applications with 32-bit Emulators

If you previously created IHLLAPI applications for use with 16-bit (Windows 3.1x) terminal emulators, you can use those applications with the 32-bit (Windows 7, Vista, XP, and 2000) emulators with some minor modifications:

- Copy the IHLLAPI.DLL and WHLLAPI.DLL included with the 32-bit Accessory Manager over the existing IHLLAPI.DLL and WHLLAPI.DLL that you're using with your application.

  Note: When you overwrite your existing files with the new versions, you will not be able to use your IHLLAPI application with the 16-bit emulators.

  If you need to use both the 16-bit and 32-bit emulators, it is recommended that you create two folders for your application. In one, include the original IHLLAPI.DLL and WHLLAPI.DLL for use with 16-bit emulators, and in the other, include the new IHLLAPI.DLL and WHLLAPI.DLL for use with the 32-bit emulators.

- Make sure that the IHLLAPI.DLL is in the same folder as the IHLAPI32.DLL.
- If you completed the preceding procedures and experience any difficulty using your existing application, you might need to make additional modifications. Specifically, if your application was written in C, C++, or another language that includes header files, you might need to include WHLLAPI.H, HLLAPI.H, and IHLAPI32.H and recompile. In addition, if your application uses WHLLAPI.LIB, you must change this to IHLLAPI.LIB and recompile.

Using Short Names

When writing IHLLAPI applications, you must perform two tasks that involve short names:

- Associate a session file name with at least one short name
- Specify which short name to use for each IHLLAPI function

Associating Sessions with Short Names

You can associate short names with session file names using Accessory Manager or using an IHLLAPI function call from your application. The procedure for doing this using Accessory Manager is provided below. For information about doing this using an IHLLAPI function call, refer to "Associate Profile (911)" on page 147.

1. Run Accessory Manager.

   Click the Start button, point to Programs, point to INFOConnect 32-bit, and click Accessory Manager 32-bit.

2. From the Options menu, click Global Preferences.

3. Click the HLLAPI tab.

4. Click the desired short name.

   If you click a short name that has no session name after it, you can assign a session name to that short name. If you click a short name that already has a session name after it, you can change the session name.

5. Click Browse and double-click the session to associate with that short name.

   Note: To remove a session name from a short name, click the short name and click Remove.

Specifying Short Names

Many IHLLAPI functions require a short name in their calling parameters. However, hard-coding a short name can cause several problems:

- The corresponding session might be in use by the user or another application.
- The same short name might be used for one session type on one PC and a different session type on another PC.

  For example, the short name A might be associated with a T 27 session on one PC and a UTS session on another PC. Errors occur if you try to use an application designed for use with one type of emulator with a different emulator.

To avoid these problems, use the Get Associations (913) function to retrieve a list of all short names that have been associated with sessions, then use the Query Sessions (10) function to determine which sessions are already open. By comparing these lists, you can determine whether a session with a short name is not already in use. Your application can then either select a session automatically or display a list for the user to choose from.

The Query Sessions (10) and Query Session Status (22) functions both provide information about what type of terminal is being emulated by the session. You can use these functions to determine whether a particular short name is associated with the right type of session for use with your application.

Examples

The following pseudo-code examples show how to obtain and use short names. To keep the examples short and simple, normal programming considerations such as waits and full return code handling have been omitted. In addition, these examples show how to connect to an active (already open) session.

Example 1

This example shows how to use Query Sessions (10) to obtain a short name that can be used to connect to a particular session type's presentation space.

```
Define global variable SHORT_NAME for session short
  name
Start Sequence

  Call Query Sessions (10) function
    Returns: Variable string containing
    information on all the open sessions.
  End Query Sessions

  Search for first host session
    This is typically a loop that searches for the
    first occurrence of a session of a specific
    type (H, A, O, or U), and then exits the loop
    upon finding it.
    Returns:  First valid short name returned and
    placed in global variable SHORT_NAME.
  End Search

  Call Connect Presentation Space (1) function
    Pass the global variable SHORT_NAME as the
    DataString of this function.
    Returns:  Success/failure status.
  End Connect

End of Sequence
```

Example 2

This example adds the Query Session Status (22) function, which examines the available sessions for various characteristics and presentation space size. Here, the presentation space size is stored in a global variable to be used elsewhere by the application.

```
Define global variable NAME for short name
Define global variable ROW for number of rows in PS
Define global variable COL for number of columns in PS

Start Sequence

   Call Query Sessions (10) function
      Returns:  Variable string containing
      information on all the open sessions.
   End Query Sessions

   Search for first host session
      This is a loop that searches for the first
      occurrence of a session of a specific type
      (H, A, O, or U), and then exits the loop upon
      finding it.
      Returns:  First valid short name returned and
      placed in global variable NAME.
   End Search

   Call Query Session Status (22) function
      Pass the global variable NAME as part of the
      DataString of this function.
      Returns:  Variable string containing the
      session short name, session type, session
      characteristics, and number of rows and
      columns in the presentation space.
   End Query Session Status

   Read Data String
      Assign global variable ROW from bytes 12 and 13
      of the returned DataString.
      Assign global variable COL from bytes 14 and 15
      of the returned DataString.
   End Read

   Call Connect Presentation Space (1) function
      Pass the global variable NAME as the
      DataString of this function.
      Returns:  Success/failure status.
   End Connect

End of Sequence
```

Connecting to Multiple Sessions

You can write IHLLAPI applications that use multiple sessions. For example, you can write an application that communicates with both a T 27 session and a UTS session.

However, IHLLAPI works with only one session at a time. To work with multiple sessions, you must use the following procedure:

1. Use the Connect Presentation Space (1) function to connect to one session (such as an T 27 session).
2. Issue the appropriate function calls for that session.
3. Use the Disconnect Presentation Space (2) function to disconnect from that session.
4. Repeat steps 1 through 3 for each session that you want to use.

Setting Session Parameters

Using the Set Session Parameters (9) function, you can set session parameters that determine how certain IHLLAPI functions operate.

When designing your IHLLAPI application, keep the following guidelines in mind:

- Call the Reset System (21) function at the beginning of your application to reset all session parameters to their default settings. This ensures that your application operates in a known environment.
- Call the Set Session Parameters (9) function to set the session options before connecting to the session or immediately after connecting.

  Using the proper options can make your application more efficient. For example, when the STREOT session parameter is set, your application will simply look for an end-of-text character rather than calculate and specify the length of strings explicitly.
- Call the Reset System (21) function before closing your application.

For more information on the Set Session Parameters (9) function, refer to "Set Session Parameters (9)" on page 71.

Interacting with Host Applications

When users interact directly with host applications, they can rely on visual cues to determine which host application screen is displayed, where the cursor is, which mode the session is operating in, how the host has responded to commands or data, and other similar information.

IHLLAPI applications must rely on function calls to obtain this kind of data and respond accordingly. For example, to determine which host screen is displayed, the IHLLAPI application might issue a function call to search for a particular string on the screen.

When interacting with host applications, use the following IHLLAPI functions:

Function	Description
Connect Presentation Space (1)	Connect to a session and check the status of that session
Search Presentation Space (6)	Search the presentation space for host messages or prompts
Copy String to Presentation Space (15) or Copy String to Field (33)	Send data to a presentation space position or field
Send Key (3)	Transmit data or a command to the host
Wait (4) or Pause (18)	Wait for the host to process the data or command
Search Presentation Space (6), Copy Presentation Space to String (8), or Copy OIA (13)	Determine how the host responded
Disconnect Presentation Space (2)	Disconnect your application from the session

For more information about optimizing your application's interactions with the host, refer to "Waiting for Host Responses" on page 27. For more information about all these functions, refer to Chapter , "IHLLAPI Functions."

Waiting for Host Responses

After your application sends data or a command to the host, it must wait for the host to complete processing the data or otherwise respond to the command.

There are several ways to allow for this. For example, you can make your application pause for a specific amount of time, or have your application periodically check for an indication that the session is ready to accept additional input.

However, each of these methods does have disadvantages. For example, a timed pause might not allow sufficient time if the load on the host, the network, or the PC processor causes unexpected delays. Conversely, a timed pause might also cause your application to operate more slowly than needed, allowing more time than is actually required for processing.

Periodically checking the session for readiness can also cause problems. If your application continuously checks for a host response, system resources might not be available for other applications running on the PC.

The optimum technique to use for waiting for host responses varies depending on the type of programming language you use, the PCs used to run the application, the type of host the PCs communicate with, and the type of network connection between the PC and the host.

The following table lists several IHLLAPI functions that you can use to incorporate delays for host responses in your application.

Function	Session Parameter	Description
Wait (4)	NWAIT	Check the session status and return information about it immediately Adding a timed delay and counter variables and using this setting provides the simplest method of waiting for a host response.
	TWAIT	Wait up to one minute for the keyboard to become ready for input, then return information about the session status to your application
Pause (18)	FPAUSE	Pause for the length of time specified in the Pause (18) function
	IPAUSE	Pause until the Start Host Notification (23) function returns a value indicating that a host event has occurred
Query Host Update (24)	Not applicable	Determine whether the presentation space or OIA has been updated since Start Host Notification (23) was called or this function was previously called
Search Presentation Space (6) or Search Field (30)	Not applicable	Search for a particular string that indicates a host response (such as the appearance of a new host application screen)
Copy OIA (13)	Not applicable	Determine whether WAIT, XMIT, or similar information that indicates that the session cannot accept input still appears in the status line

Timer and Delay Routines

The PC processor's speed, sometimes referred to as clock speed, determines the length of time it takes for the CPU to execute an instruction. Normally, faster processors are advantageous because by increasing the instruction execution speed, the application runs faster. However, in an IHLLAPI application this can become a disadvantage, especially when you create timers or delays to wait for events that occur on a host.

Most problems occur when FOR-NEXT loops or incremental counters are used to create a timed delay. These routines depend on the efficiency of the compiler and on the processor speed of the PC they are executed on. If an application was developed on a slow PC, these loops might not provide adequate time delays on a faster PC; if the application was developed on a fast PC, it has long delays on a slow PC.

Timer and delay routines should be based on constant real-time events such as the PC's clock tick or the time-of-day information (system clock). The PC's clock tick occurs 18.2 times per second; the system clock ticks at one-second intervals. These real time events are constant across all models and processor speeds of PCs, so they provide a consistent foundation for timers and delays.

For more detailed information on the PC's clock tick and time-of-day information, consult your Microsoft Windows SDK documentation.

Using Field Functions

IHLLAPI provides several functions that work with fields:

- Query Field Attribute (14)
- Search Field (30)
- Find Field Position (31)
- Find Field Length (32)
- Copy String to Field (33)
- Copy Field to String (34)

Field Function Limitations

Field functions provide a finer degree of control than functions that use the entire presentation space. However, you must keep the following considerations in mind:

- **Field functions are time-consuming operations.** Each call to these functions allocates and frees memory, extracts whole screen information (including attributes), processes the screen information to identify the field structures, calculates the field extents, and then performs the specified function. For example, if your application calls Find Field Position (31), Find Field Length (32), and Copy String to Field (33), the IHLAPI32.DLL completes the whole processing three times.
- **Unformatted presentation spaces do not support field function calls.** Because there are no individually defined fields in an unformatted presentation space, you should not use field functions with unformatted presentation spaces.

Where processing time is a consideration or when you are working with unformatted presentation spaces, you should use functions that work with the entire presentation space rather than field functions. For example, you can use Search Presentation Space (6) rather than Search Field (30).

Working with Fields

When working with fields in a formatted presentation space, your application must be able to perform the following tasks:

- Determine the location of each field
- Determine the characteristics of each field
- Send the proper keystrokes to the correct location

Failure to take these conditions into account can cause your application to send erroneous data to the host. For example, when sending data to a field, your application should verify whether data already exists in that field. If data exists, clear the field before sending new data. This ensures that the new data is not appended to the existing data.

In addition, even small changes in a host application can cause errors to occur in your application. For example, if your application expects to find a particular string in the presentation space, an error would occur if that string changed due to changes in the host application. In this case, you must change your application to match the host application.

Be sure to design your application so that you can easily modify it to conform to changes in the host application. For example, if your application expects to find a particular string in the presentation space, put that string in a resource file so that you can quickly access and change it as needed.

Handling Return Codes and Errors

Each IHLLAPI function uses a return code to indicate the result of the execution of that function. For example, the Disconnect Presentation Space (2) function returns one of the following codes:

Return Code	Description
WHLLOK (0)	The function was successful; your application disconnected from the presentation space.
WHLLNOTCONNECTED (1)	Your application is not connected to a session.
WHLLSYSERROR (9)	The function failed due to a system error.

Some errors can be prevented by ensuring that functions are called in the correct order. For example, to avoid the WHLLNOTCONNECTED (1) error in the preceding example, you must call the Connect Presentation Space (1) function before calling the Disconnect Presentation Space (2) function. Refer to the prerequisites for each function described in Chapter , "IHLLAPI Functions," to ensure that your application avoids generating errors that can be prevented.

Others errors can be corrected by your application. Your application must manage these return codes properly to keep it running smoothly, regardless of the PC's processor speed, network load, or host response speed.

When writing an error handling routine, remember to save the call parameters before calling the function. The PSPosition_ReturnCode in the call parameter always changes in the return parameter. (Other call parameters might change as well, depending on the function.) If your error handler tries to correct a problem and then call the function again, the handler must be able to find or reconstruct the original call parameters passed to the function.

For a list of the standard return codes, their causes, and suggestions for handling them, refer to Appendix A, "Return Codes." For details about the return parameters for a particular function, refer to Chapter , "IHLLAPI Functions."

Developing IHLLAPI Applications in Visual Basic

For Visual Basic applications to access the IHLLAPI functions, you must include the following declarations and syntax.

Variable Declarations

You must use the following declarations in your application:

Variable Declaration	Explanation
ApiFunc	This integer passes the requested IHLLAPI function number.
ApiString	This string passes or receives a data string needed by some IHLLAPI functions.
ApiLen	This integer passes the length of the string variable or receives integer data from some IHLLAPI functions.
ApiRetc	This integer passes data to some IHLLAPI functions and receives a return code back from all IHLLAPI functions.

In addition, you must declare the IHLLAPI entry point in your application's global module, as follows:

```
Declare Sub WinHLLAPI Lib "drive:\dir\IHLAPI32.DLL" _
(ApiFunc As Integer, ByVal ApiString As String, _
ApiLen As Integer, ApiRetc As Integer)
```

This declaration is provided in the VBHLLAPI.BAS file included with the ADK. You can include this file in your project.

Call Function Syntax

When writing a call, you must use the following syntax:

```
CALL WinHLLAPI(ApiFunc, ApiString, ApiLen, ApiRetc)
```

Developing IHLLAPI Applications in C

For your C application to have access to the IHLLAPI functions, you must include the following function declaration and variables in either the main body of your application or, where appropriate, in a header file that is included when you compile the application.

Header Files

The Accessory Development Kit comes with three header files (WHLLAPI.H, HLLAPI.H, and IHLAPI32.H) that include several function declarations that must be included in your project.

For a listing of these files, refer to Appendix C, "IHLLAPI Header Files."

Variable Declaration

You must use the following declarations in your application:

Variable Declaration	Explanation
LPWORD *FunctionNumber*	A pointer to the defined function name of the IHLLAPI function being called. It has a corresponding constant, included in IHLAPI32.H (refer to Appendix , "IHLLAPI Header Files," for a listing). For example, when using the Connect Presentation Space (1) function, the value sent would be CONNECTPS.
LPBYTE *DataString*	A pointer to the data string. Although used by most IHLLAPI functions, not all functions require a data string. Some functions use the data string only on the call, some only on the return, and some on both the call and the return. Refer to the function descriptions in Chapter 3, "IHLLAPI Functions," for details about the requirements of each function.
LPWORD *DataLength*	A pointer to the data length or the length of the data string, depending on the IHLLAPI function. Some functions use the data length only on the call some only on the return, and some on both the call and the return.
LPWORD *PSPosition_ReturnCode*	A pointer to the presentation space position/ return code. This parameter supplies data to the IHLAPI32.DLL on the call, and supplies the response from the application on the return.

Running Your IHLLAPI Application

When running your IHLLAPI application, verify that the following criteria are met:

- The IHLAPI32.DLL is in the Windows PATH.
- Accessory Manager is running and a session is open.

  You can either run Accessory Manager and open a session manually, or you can run it using the Connect Presentation Space (1) function.

  For instructions on using Accessory Manager, refer to the online Help for that product.

Windows automatically loads and removes the IHLAPI32.DLL and all supporting software as required.

IHLLAPI Functions 3

In This Chapter

This chapter includes the following headings. It also includes a heading for each IHLLAPI function in numerical order.

Unsupported HLLAPI Functions

The following table lists the WinHLLAPI and EHLLAPI functions that are not supported in this version of IHLLAPI:

Number	Function
11	Reserve
12	Release
17	Storage Manager
41	Start Close Intercept
42	Query Close Intercept
43	Stop Close Intercept
60	Lock Presentation Space API
61	Lock Window Services API
105	Change Switch List LT Name
106	Change Presentation Space Window Name

How IHLLAPI Functions Are Documented

The description of each function in this chapter includes the following information:

- Purpose
- Prerequisites
- Syntax
- Call parameters
- Return parameters
- Comments
- See Also

Purpose

Immediately following the function name is a brief description of the function and how it should be used.

Prerequisites

Some functions do not operate correctly unless the application calls other functions first. This section lists the functions that your application must call before calling the function being explained.

Syntax

The syntax for all IHLLAPI calls is always the same:

```
WinHLLAPI(FunctionNumber,DataString,DataLength,
   PSPosition_ReturnCode)
```

The following table describes each parameter:

Parameter	Description
FunctionNumber	A pointer to an integer
DataString	A pointer to an array of bytes
DataLength	A pointer to an integer
PSPosition_ReturnCode	A pointer to an integer

When your application calls the IHLAPI32.DLL using this syntax, it specifies the appropriate value for each parameter as described in "Call Parameters." When the IHLAPI32.DLL returns the results of the call to the application, it uses the same syntax but replaces the original call parameter values with the appropriate return parameter values described in "Return Parameters."

Call Parameters

This section describes the parameters your application must pass to the IHLAPI32.DLL.

Note: Use zeros or null strings for any unused parameters.

Return Parameters

This section describes the parameters that the IHLAPI32.DLL returns to your application as a result of the function call.

If any call parameters are not replaced by return parameters, the original call parameters are returned unchanged.

All functions pass a return code as a return parameter. Many functions use standard return codes, but certain functions interpret the return codes differently, or use additions to the standard return codes. Refer to Appendix A, "Return Codes," and the function descriptions in this chapter for more information about return codes.

Comments

This section describes special considerations and restrictions, as well as how to use the function with other IHLLAPI functions. This section also points out how to use the function with different terminal emulators.

See Also

This section lists other related IHLLAPI functions.

Alphabetical Function List

The following tables provide an alphabetical list of the IHLLAPI functions:

Function	Number
Associate Profile	911
Connect Presentation Space	1
Connect Window Services	101
Convert Position or RowCol	99
Copy Field to String	34
Copy OIA	13
Copy Presentation Space	5
Copy Presentation Space to String	8
Copy String to Field	33
Copy String to Presentation Space	15
Disconnect Presentation Space	2
Disconnect Window Services	102
Find Field Length	32
Find Field Position	31
Find File Name	914
Find Short Name	915
Get Associations	913
Get Key	51
Pause	18
Post Intercept Status	52
Query Cursor Location	7
Query Field Attribute	14
Query Host Update	24
Query Session Status	22
Query Sessions	10
Query Sessions Full	910
Query System	20
Query Window Coordinates	103
Receive File	91
Remove Profile	912
Reset System	21
Search Field	30
Search Presentation Space	6
Send File	90
Send Key	3
Set Cursor Position	40
Set Session Parameters	9
Start Host Notification	23
Start Keystroke Intercept	50
Stop Host Notification	25
Stop Keystroke Intercept	53
Wait	4
Window Status	104

Numerical Function List

The following tables provide a numerical list of the IHLLAPI functions:

Number	Function
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
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

Number	Function
32	Find Field Length
33	Copy String to Field
34	Copy Field to String
40	Set Cursor Position
50	Start Keystroke Intercept
51	Get Key
52	Post Intercept Status
53	Stop Keystroke Intercept
90	Send File
91	Receive File
99	Convert Position or RowCol
101	Connect Window Services
102	Disconnect Window Services
103	Query Window Coordinates
104	Window Status
910	Query Sessions Full
911	Associate Profile
912	Remove Profile
913	Get Associations
914	Find File Name
915	Find Short Name

Connect Presentation Space (1)

The Connect Presentation Space function connects your IHLLAPI application to the specified presentation space. After your application connects to the presentation space, all communication with the host occurs through it.

Prerequisites

You must associate a session file name (.ADP) with a short name using either Accessory Manager or the Associate Profile (911) function.

Syntax

```
WinHLLAPI(FunctionNumber,DataString,DataLength,
   PSPosition_ReturnCode)
```

Call Parameters

Parameter	Value
FunctionNumber	CONNECTPS (1)
DataString	The short name of the session to connect with.
DataLength	Not applicable; a length of 1 byte is implied.
PSPosition_ReturnCode	Not applicable.

Return Parameters

Parameter	Value	Description
DataString	Not applicable.	Not applicable.
DataLength	Not applicable.	Not applicable.

Parameter	Value	Description
PSPosition_ReturnCode	WHLLOK (0)	Function succeeded.
	WHLLNOTCONNECTED (1)	An invalid short name was specified.
	WHLLPSBUSY (4)	The connection was successfully made, but the presentation space is busy or input is inhibited.
	WHLLINHIBITED (5)	The connection was successful, but the keyboard is locked.
	WHLLSYSERROR (9)	The function failed due to a system error.
	WHLLUNAVAILABLE (11)	The specified presentation space is already in use.

Comments

This function runs Accessory Manager in a hidden state (if it is not already running), opens the specified session (if it is not already open), and connects to that session.

If the CONLOG session parameter is set (which is the default state), your application window remains the active window on the desktop.

If the CONPHYS session parameter is set, the terminal emulation session window becomes the active window on the desktop.

See Also

Disconnect Presentation Space (2), Associate Profile (911)

Disconnect Presentation Space (2)

The Disconnect Presentation Space function disconnects your application from the current presentation space.

Prerequisites

Connect Presentation Space (1)

Syntax

```
WinHLLAPI(FunctionNumber,DataString,DataLength,
   PSPosition_ReturnCode)
```

Call Parameters

Parameter	Value
FunctionNumber	DISCONNECTPS (2)
DataString	Not applicable.
DataLength	Not applicable.
PSPosition_ReturnCode	Not applicable.

Return Parameters

Parameter	Value	Description
DataString	Not applicable.	Not applicable.
DataLength	Not applicable.	Not applicable.
PSPosition_ReturnCode	WHLLOK (0)	Function succeeded.
	WHLLNOTCONNECTED (1)	Your application is not connected to a session.
	WHLLSYSERROR (9)	The function failed due to a system error.

Comments

If the DISCONPHYS session parameter is set (which is the default state), this function disconnects your application from the presentation space, disconnects the session from the host, and closes the session.

If the DISCONLOG session parameter is set, this function disconnects your application from the presentation space, but does not close the session.

As part of your application termination routine, always issue a Reset System (21) call. This resets all the session parameters to their defaults and helps avoid problems with any subsequent use of the presentation space.

After the application disconnects from the presentation space, functions that require a connected presentation space return an error.

See Also

Connect Presentation Space (1), Set Session Parameters (9), Reset System (21)

Send Key (3)

The Send Key function places a keystroke or string of keystrokes in the presentation space at the current cursor location. This function is equivalent to typing on the keyboard.

Prerequisites

Connect Presentation Space (1)

Syntax

```
WinHLLAPI(FunctionNumber,DataString,DataLength,
   PSPosition_ReturnCode)
```

Call Parameters

Parameter	Value
FunctionNumber	SENDKEY (3)
DataString	String of keystrokes (up to 255 bytes). Refer to the tables later in this section for valid keystrokes.
DataLength	Number of bytes in the DataString.
PSPosition_ReturnCode	Not applicable.

Return Parameters

Parameter	Value	Description
DataString	Not applicable.	Not applicable.
DataLength	Not applicable.	Not applicable.

Parameter	Value	Description
PSPosition_ReturnCode	WHLLOK (0)	Function succeeded.
	WHLLNOTCONNECTED (1)	Your application is not connected to a session.
	WHLLPARAMETERERROR (2)	An incorrect call parameter was passed.
	WHLLPSBUSY (4)	The presentation space is busy or input was inhibited; some keystrokes were not sent.
	WHLLINHIBITED (5)	Input to the presentation space was inhibited or rejected; some keystrokes were not sent.
	WHLLSYSERROR (9)	The function failed due to a system error.

Comments

Refer to "Waiting for Host Responses" on page 27 for suggested methods for waiting for the session to become available before sending keystrokes to the presentation space.

To send terminal keystrokes, you can use either the terminal keystroke name enclosed in angle brackets or an escape character (@ is the default) followed by an ASCII mnemonic code. For example, to send the transmit keystroke, you can use either <Transmit> or @E. The ASCII mnemonic codes are listed in the following tables.

To send T 27 control sequences using the ASCII mnemonics, send the code for Ctrl (@A@^) followed by the desired character. For example, to send Ctrl w, the *DataString* would be @A@^w.

The ASCII mnemonics apply to the total length of the *DataString* (which cannot exceed 255 characters). For example, if you send the code for transmit (@E), you must include two bytes in the *DataLength* parameter.

See Also

Connect Presentation Space (1), Set Session Parameters (9)

ASCII Mnemonic Codes

The following tables list the ASCII mnemonic codes for T 27, UTS, and VT terminal keystrokes.

You can change the escape character using the Esc=c session parameter. Refer to "Set Session Parameters (9)" on page 71 for more information.

T 27 Terminal Keystrokes

Terminal Keystroke	Function	ASCII Mnemonic
BackSpace	Moves the cursor one column to the left without deleting any characters If the cursor is in the first column of a line, it moves to the last column of the preceding line. If it is home, it moves to the last column of the last line of the page.	@<
CarriageReturn	Moves the cursor to the first column of the next line or the first column of the current line, depending on the session configuration	@N
ClearPageCursorHome	Deletes all text on the page and moves the cursor home In forms mode, this keystroke clears either text in unprotected fields or all text, depending on the session configuration. Subsequently, the cursor moves to the first unprotected field.	@C
ClearToEndOfLine	Deletes all text from the cursor to the end of the line In forms mode, this keystroke deletes all text from the cursor to the end of the field.	@F
CTRL	Puts the session in control mode	@A@^
CursorToEndOfPage	Moves the cursor to the last position on the current page	@q
DeleteFromLine	Deletes the character that the cursor is on and shifts the remaining characters on the line to the left	@D
DeleteLine	Deletes the line containing the cursor	@A@D
Down	Moves the cursor down one line If the cursor is in the bottom line of a page, it moves to the top line.	@V
F1...F9	Perform host-defined functions	@1...@9
F10	Performs host-defined function	@a
Home	Moves the cursor home	@0 (zero)
Ins	Toggles between overwrite mode and insert-in-line mode This keystroke might also insert a space at the cursor location, depending on the session configuration.	@A@I

T 27 Terminal Keystrokes, continued

Terminal Keystroke	Function	ASCII Mnemonic
InsertLine	Inserts a line at the cursor location, moving all subsequent lines down and moving the cursor to the first column of the new line. Any data that was on the last line of the page is lost.	@A@L
Left	Moves the cursor one column to the left If the cursor is in the first column of a line, it moves to the last column of the preceding line. If it is in the first column of the first line of the page, it moves to the last column of the last line of the page.	@L
Local	Puts the session in local mode	@A@W
PageDown	Displays the next page If the last page already is displayed, this keystroke displays the first page.	@v
PageUp	Displays the previous page If the first page already is displayed, this keystroke displays the last page.	@u
PutETX	Inserts an end-of-text character at the cursor position and moves the cursor home In insert-in-line or insert-in-page mode, characters after the ETX character shift to the right. In overwrite mode, the ETX character replaces any character at the cursor position.	@A@S
Receive	Puts the session in receive mode	@A@M
Right	Moves the cursor one column to the right If the cursor is in the last column of a line, it moves to the first column of the next line. If it is in the last column of the last line of the page, it moves to the first column of the first line of the page.	@Z
SHIFTF1	Performs host-defined function	@b
SHIFTF2	Performs host-defined function	@c
SHIFTF3	Performs host-defined function	@d
SHIFTF4	Performs host-defined function	@e
SHIFTF5	Performs host-defined function	@f
SHIFTF6	Performs host-defined function	@g
SHIFTF7	Performs host-defined function	@h
SHIFTF8	Performs host-defined function	@i

T 27 Terminal Keystrokes, continued

Terminal Keystroke	**Function**	**ASCII Mnemonic**
SHIFTF9	Performs host-defined function	@j
SHIFTF10	Performs host-defined function	@k
Specify	Sends the cursor location to the host This keystroke might also send the page number, depending on the session configuration. The session configuration also determines whether the data is sent in hexadecimal or ASCII format.	@A@H
Tab	Moves the cursor to the next tab stop If tab stops were not set, this keystroke moves the cursor home. In forms mode, this keystroke moves the cursor to the next unprotected field.	@T
TabBack	Moves the cursor to the previous tab stop If tabs were not set, this keystroke moves the cursor home. In forms mode, this keystroke moves the cursor to the previous unprotected field.	@B
Transmit	Sends data to the host Depending on the session configuration, this keystroke sends the data from home to the cursor position or from home to the end of the page.	@E
TransmitLine	Sends to the host the data from the beginning of the line to the cursor location	@A@x
Up	Moves the cursor up one line in the same column If the cursor is in the top line of a page, it moves to the bottom line.	@U

UTS Terminal Keystrokes

Terminal Keystroke	**Function**	**ASCII Mnemonic**
BackSpace	Moves the cursor one column to the left If the cursor is in the first column of a row, it moves to the last column of the preceding row. If the cursor is home, it moves to the last column of the last row of the page.	@<
CarriageReturn	Moves the cursor to the first column of the next row	@N
ClearChange	Resets the changed-field indicator to zero on all FCC fields, thereby indicating that all the fields are unchanged and should not be transmitted	@A@C
ClearPageCursorHome	Deletes all text and FCC fields in the current page and moves the cursor home	@C
ClearToEndOfField	Deletes all unprotected characters in an FCC field from the cursor to the end of the FCC field	@F
ClearToEndOfLine	Deletes all text from the cursor to the end of the row or the end of an FCC field, whichever comes first	@A@l
ClearToEndOfPage	Deletes all text from the cursor to the end of the page except protected fields	@A@p
ControlPageToggle	Displays/hides the Control Page or Extended Control Page, depending on how the session is configured	@A@^
CursorToEndOfField	Moves the cursor to the end of the FCC field If the last position is the trail byte of a DBCS character, the cursor moves to the lead byte. If the last position is protected, the cursor moves to the next unprotected character.	@A@)
CursorToEndOfLine	Moves the cursor to the last position of the current row If the last position is the trail byte of a DBCS character, the cursor moves to the lead byte. If the last position is protected, the cursor moves to the next unprotected character.	@A@]
CursorToEndOfPage	Moves the cursor to the last position on the current page If the last position is the trail byte of a DBCS character, the cursor moves to the lead byte. If the last position is protected, the cursor moves to the next unprotected character.	@q

UTS Terminal Keystrokes, continued

Terminal Keystroke	Function	ASCII Mnemonic
CursorToEOPAndXmit	Moves the cursor to the end of the page and transmits the screen	@A@T
CursorToStartOfField	Moves the cursor to the first position in the FCC field in which cursor is located If the first position is the trail byte of a DBCS character, the cursor moves back to the lead byte. If the first position is protected, the cursor moves to the next unprotected character.	@A@(
CursorToStartOfLine	Moves the cursor to the first position on the same row If the first position is the trail byte of a DBCS character, the cursor moves back to the lead byte. If the first position is protected, the cursor moves to the next unprotected character.	@A@[
DeleteFromLine	Deletes the character that the cursor is on, shifts the remaining characters on the row to the left, and inserts a space in the last column of the row In an FCC field, this keystroke deletes the character, shifts the remaining characters in that field to the left, and inserts a space in the last column of the field. Emphasis is unchanged, and the characters in the rest of the row aren't shifted.	@D
DeleteFromPage	Deletes the character that the cursor is on and shifts all the remaining characters one position to the left If the cursor is in an FCC field, the characters in that field are shifted to the left, but the characters beyond the current field aren't shifted. A space is inserted in the last position of the page or field.	@A@%
DeleteLine	Deletes the row containing the cursor Subsequent rows move up one row. The bottom row is filled with spaces.	@A@D
Down	Moves the cursor down one row in the same column If the cursor is in the bottom row of a page, it moves to the top row. If there is a protected character below, the cursor moves to the next unprotected character to the right of the protected character.	@V
DuplicateLine	Copies the row containing the cursor to the row below, overwriting whatever was on that row The cursor moves to the same column on the duplicated row.	@A@$
F1 ... F9	Perform host-defined functions	@1...@9

UTS Terminal Keystrokes, continued

Terminal Keystroke	Function	ASCII Mnemonic
F10	Performs host-defined function	@a
F11	Performs host-defined function	@b
F12	Performs host-defined function	@c
F13	Performs host-defined function	@d
F14	Performs host-defined function	@e
F15	Performs host-defined function	@f
F16	Performs host-defined function	@g
F17	Performs host-defined function	@h
F18	Performs host-defined function	@i
F19	Performs host-defined function	@j
F20	Performs host-defined function	@k
F21	Performs host-defined function	@l
F22	Performs host-defined function	@m
FCCClear	Deletes all text in the FCC field at the cursor location If there isn’t an FCC at the cursor location, this keystroke clears the closest FCC to the left of the cursor.	@A@\
FCCEnable	Re-enables FCCs so that you can enter data in them	@A@e
FCCGenerate	Initiates the FCC definition process using the keyboard instead of the Generate FCC dialog box	@A@g
FCCLocate	Moves the cursor to the first character of the next FCC, whether it is protected or not If this character is protected, protection is cleared.	@A@/
Home	Moves the cursor home If the home position is protected, this keystroke moves the cursor to the first unprotected field on the page.	@0 (zero)

UTS Terminal Keystrokes, continued

Terminal Keystroke	**Function**	**ASCII Mnemonic**
InsertInPage	Inserts a space at the cursor location, moving subsequent characters on the page one column to the right Any character in the rightmost column moves to the first position on the next row. If the cursor is in an FCC field, the subsequent characters in that field are shifted to the right, but the characters in the rest of the page aren't shifted. If there is a character in the last column in the page or field, it is lost.	@A@*
InsertLine	Inserts a row at the cursor location and shifts subsequent rows down one The last row on the page is lost.	@A@L
Left	Moves the cursor one column to the left If the cursor is in the first column of a row, it moves to the last column of the preceding row. If it is in the first column of the first row of the page, it moves to the last column of the last row of the page. If there is a protected character to the left, the cursor moves to the next unprotected character to the right of the protected character.	@L
MessageWait	Displays the waiting host message	@A@M
PageDown	Displays the next page If the last page already is displayed, this keystroke displays the first page.	@v
PageUp	Displays the previous page If the first page already is displayed, this keystroke displays the last page.	@u
Right	Moves the cursor one column to the right If the cursor is in the last column of a row, it moves to the first column of the next row. If it is in the last column of the last row of the page, it moves to the first column of the first row of the page. If there is a protected character to the right, the cursor moves to the next unprotected character to the left of the protected character.	@Z

UTS Terminal Keystrokes, continued

Terminal Keystroke	**Function**	**ASCII Mnemonic**
SetStartOfEntry	Inserts a start-of-entry (SOE) character at the cursor position In insert mode, characters after the SOE character shift to the right. In overwrite mode, the SOE character replaces any character at the cursor position.	@A@S
SystemMode	Puts the session in system mode so that you can send and receive certain commands to and from a System 80 host	@A@Q
Tab	Moves the cursor to the next tab stop (either an FCC tab or a tab on the screen) If tabs were not set, this keystroke moves the cursor home.	@T
TabBack	Moves the cursor to the previous tab stop (either an FCC tab or a tab on the screen) If tabs were not set, this keystroke moves the cursor home.	@B
TabSet	Places a tab on the screen at the cursor location and moves the cursor one column to the right	@A@=
Transmit	Sends data to the host Depending on the session configuration, this keystroke sends either all data, only unprotected data, or only changed data.	@E
UnlockKeyboard	Restores keyboard functionality when it is locked due to a communication error	@R
Up	Moves the cursor up one row in the same column If the cursor is in the top row of a page, it moves to the bottom row. If there is a protected character above, the cursor moves to the next unprotected character to the right of the protected character.	@U
WorkstationMode	Exits system mode and restores the session page	@A@W

VT Terminal Keystrokes

Terminal Keystroke	**ASCII Mnemonic**
Backspace	@<
Break	@A@Q
Down	@V
Enter	@E
F6	@6
F7	@7
F8	@8
F9	@9
F10	@a
F11	@b
F12	@c
F13	@d
F14	@e
F15	@f
F16	@g
F17	@h
F18	@i
F19	@j
F20	@k
Insert	@A@I
Left	@L
Next	@v
PF1	@1
PF2	@2
PF3	@3
PF4	@4
Prev	@u
Remove	@D
Right	@Z
Up	@U

Wait (4)

The Wait function checks the status of the session to determine whether it can accept input.

Prerequisites

Connect Presentation Space (1)

Syntax

```
WinHLLAPI(FunctionNumber,DataString,DataLength,
   PSPosition_ReturnCode)
```

Call Parameters

Parameter	Value
FunctionNumber	WAIT (4)
DataString	Not applicable.
DataLength	Not applicable.
PSPosition_ReturnCode	Not applicable.

Return Parameters

Parameter	Value	Description
DataString	Not applicable.	Not applicable.
DataLength	Not applicable.	Not applicable.
PSPosition_ReturnCode	WHLLOK (0)	The session is ready for input.
	WHLLNOTCONNECTED (1)	Your application is not connected to a session.
	WHLLPSBUSY (4)	The presentation space is busy or input is still inhibited.
	WHLLINHIBITED (5)	The keyboard is locked and input is inhibited.
	WHLLSYSERROR (9)	The function failed due to a system error.

Comments

Since keystrokes are not accepted while the session is waiting for a response from the host, use the Wait (4) function to check the status of the session before sending keystrokes using the Send Key (3) function.

If the TWAIT session parameter is set (which is the default state), the Wait (4) function waits for up to one minute before returning the appropriate return parameter to the application. If the session becomes ready for input within that time, the function returns WHLLOK (0) as soon as the session is ready. If the session does not become ready for input within that time, the function returns WHLLINHIBITED (5) at the end of the minute.

If the NWAIT session parameter is set, this function does not wait; it returns the current status of the session immediately.

For more information about session parameters, refer to "Set Session Parameters (9)" on page 71.

Refer to "Waiting for Host Responses" on page 27 for other methods for waiting for the session to become available before sending keystrokes to the presentation space.

See Also

Connect Presentation Space (1), Send Key (3), Set Session Parameters (9)

Copy Presentation Space (5)

The Copy Presentation Space function copies the contents of the presentation space to a string in your application.

Prerequisites

Connect Presentation Space (1)

Syntax

```
WinHLLAPI(FunctionNumber,DataString,DataLength,
   PSPosition_ReturnCode)
```

Call Parameters

Parameter	Value
FunctionNumber	COPYPS (5)
DataString	A data string defined in your application that will contain the data in the presentation space. If the EAB session parameter is set, this string must be two or three times the size of the presentation space (see “Comments”).
DataLength	Not applicable (the length of the presentation space is implied).
PSPosition_ReturnCode	Not applicable.

Return Parameters

Parameter	Value	Description
DataString	A string containing the contents of the presentation space.	See “Comments” for information about how the data is returned.
DataLength	Not applicable.	Not applicable.
PSPosition_ReturnCode	WHLLOK (0)	Function succeeded.
	WHLLNOTCONNECTED (1)	Your application is not connected to a session.
	WHLLINHIBITED (5)	The copy was successful, but the keyboard is locked.
	WHLLSYSERROR (9)	The function failed due to a system error.

Comments

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

If the NOATTRB session parameter is set (which is the default state), bytes that are less than 0x1F are translated to spaces (0x20). If the ATTRB session parameter is set, all bytes are passed as their original values.

If the NOEAB session parameter is set (which is the default state), this function copies only the data in the presentation space; it does not copy any extended attributes.

If the EAB session parameter is set, each character in the returned string is followed immediately by the attribute information for that character. Therefore, you must allocate a *DataString* that is long enough to accommodate both the screen text and the attribute information.

For T 27, UTS 20, and UTS 40 sessions, the *DataString* must be twice the size of the presentation space (one character and one character attribute byte for each presentation space position). For example, if the presentation space includes 1920 presentation space positions, the *DataString* must be 3840 bytes. For UTS 60 sessions, the *DataString* must be three times the size of the presentation space (one character, one character attribute byte, and one color attribute byte for each presentation space position).

For information about the attribute bytes returned by T 27 and UTS, refer to Appendix B, “Attribute Values.”

See Also

Connect Presentation Space (1), Copy Presentation Space to String (8), Set Session Parameters (9), Copy Field to String (34)

Search Presentation Space (6)

The Search Presentation Space function scans the current presentation space for a specified string.

Prerequisites

Connect Presentation Space (1)

Syntax

```
WinHLLAPI(FunctionNumber,DataString,DataLength,
   PSPosition_ReturnCode)
```

Call Parameters

Parameter	Value
FunctionNumber	SEARCHPS (6)
DataString	The string that you want this function to search for. If the STREOT session parameter is set, the last character in this string must be an end-of-text character.
DataLength	If the STRLEN session parameter is set (which is the default state), this is the length of the DataString. If the STREOT session parameter is set, this parameter is ignored.
PSPosition_ReturnCode	If the SRCHALL session parameter is set (which is the default state), this parameter is ignored. If the SRCHFROM session parameter is set, this is the presentation space position where you want the search to begin. The upper left corner (row 1, column 1) is position 1.

Return Parameters

Parameter	Value	Description
DataString	Not applicable.	Not applicable.
DataLength	0 or greater than 0	If the value is 0, the string was not found. A value greater than 0 indicates the presentation space position where the string began.
PSPosition_ReturnCode	WHLLOK (0)	Function succeeded.
	WHLLNOTCONNECTED (1)	Your application is not connected to a session.
	WHLLPARAMETERERROR (2)	An incorrect call parameter was passed.
	WHLLPOSITIONERROR (7)	An invalid presentation space position was passed.
	WHLLSYSERROR (9)	The function failed due to a system error.
	WHLLNOFIELD (24)	The specified string was not found.

Comments

Use this function to confirm that specific data exists in the presentation space. For example, if your application expects a specific prompt before sending data to the host, use this function to search for the prompt before sending the data.

See Also

Connect Presentation Space (1), Set Session Parameters (9), Search Field (30)

Query Cursor Location (7)

The Query Cursor Location function returns the location of the cursor in the presentation space.

Prerequisites

Connect Presentation Space (1)

Syntax

```
WinHLLAPI(FunctionNumber,DataString,DataLength,
   PSPosition_ReturnCode)
```

Call Parameters

Parameter	Value
FunctionNumber	QUERYCURSORLOC (7)
DataString	Not applicable.
DataLength	Not applicable.
PSPosition_ReturnCode	Not applicable.

Return Parameters

Parameter	Value	Description
DataString	Not applicable.	Not applicable.
DataLength	The presentation space position where the cursor is located	This value is always less than or equal to the presentation space size.
PSPosition_ReturnCode	WHLLOK (0)	Function succeeded.
	WHLLNOTCONNECTED (1)	Your application is not connected to a session.
	WHLLSYSERROR (9)	The function failed due to a system error.

Comments

For information about converting the presentation space position to column and row coordinates, refer to "Send File (90)" on page 128.

Use the Set Cursor Position (40) function to change the cursor position.

See Also

Connect Presentation Space (1), Set Cursor Position (40), Convert Position or RowCol (99)

Copy Presentation Space to String (8)

The Copy Presentation Space to String function copies all or part of the presentation space into a data string defined in your application.

Prerequisites

Connect Presentation Space (1), Set Session Parameters (9)

Syntax

```
WinHLLAPI(FunctionNumber,DataString,DataLength,
   PSPosition_ReturnCode)
```

Call Parameters

Parameter	Value
FunctionNumber	COPYPSTOSTR (8)
DataString	A data string defined in your application that will contain the data in the presentation space. If the EAB session parameter is set, this string must be two or three times the size of the presentation space that is copied (see “Comments”).
DataLength	The number of characters that you want to copy.
PSPosition_ReturnCode	The presentation space position where you want to begin copying. The upper left corner (row 1, column 1) is position 1.

Return Parameters

Parameter	Value	Description
DataString	The string copied from the presentation space.	Not applicable.
DataLength	Not applicable.	Not applicable.

Parameter	Value	Description
PSPosition_ReturnCode	WHLLOK (0)	Function succeeded.
	WHLLNOTCONNECTED (1)	Your application is not connected to a session.
	WHLLPARAMETERERROR (2)	A DataLength of zero was passed.
	WHLLINHIBITED (5)	The copy was successful, but the keyboard is locked.
	WHLLPOSITIONERROR (7)	An invalid presentation space position was passed.
	WHLLSYSERROR (9)	The function failed due to a system error.

Comments

T 27 supports the display of a user message on an additional row outside of the screen size (the 25th row in a 24-row screen). This message is displayed by the host and cannot be edited by the user. However, this function can copy this row as part of the presentation space.

If the NOEAB session parameter is set (which is the default state), this function copies only the data in the presentation space; it does not copy any extended attributes.

If the EAB session parameter is set, each character in the returned string is followed immediately by the attribute information for that character. Therefore, you must allocate a *DataString* that is long enough to accommodate both the screen text and the attribute information.

For T 27, UTS 20, and UTS 40 sessions, the *DataString* must be twice the size of the presentation space (one character and one character attribute byte for each presentation space position). For example, to copy 10 characters, the *DataString* must be 20 bytes. For UTS 60 sessions, the *DataString* must be three times the size of the presentation space (one character, one character attribute byte, and one color attribute byte for each presentation space position).

For information about the attribute bytes returned by T 27 and UTS, refer to Appendix B, “Attribute Values.”

If the NOATTRB session parameter is set (which is the default state), bytes that are less than 0x1F are translated to spaces (0x20). If the ATTRB session parameter is set, all bytes are passed as their original values.

For more information about session parameters, refer to “Set Session Parameters (9)” on page 71.

See Also

Connect Presentation Space (1), Copy Presentation Space (5), Set Session Parameters (9), Copy OIA (13), Copy Field to String (34)

Set Session Parameters (9)

The Set Session Parameters function lets you specify the session parameters that control how other IHLLAPI functions operate. Unless you explicitly change the session parameters, the default session parameters apply to all functions.

Prerequisites

None.

Syntax

```
WinHLLAPI(FunctionNumber,DataString,DataLength,
   PSPosition_ReturnCode)
```

Call Parameters

Parameter	Value
FunctionNumber	SETSESSIONPARAMETERS (9)
DataString	The session parameter to set. All session parameters must be spelled exactly as listed on the following pages. If you specify multiple session parameters, be sure to separate them with either a comma or a space.
DataLength	The length of the DataString.
PSPosition_ReturnCode	Not applicable.

Return Parameters

Parameter	Value	Description
DataString	Not applicable.	Not applicable.
DataLength	Not applicable.	Not applicable.
PSPosition_ReturnCode	WHLLOK (0)	Function succeeded.
	WHLLPARAMETERERROR (2)	One or more invalid parameters were passed.
	WHLLSYSERROR (9)	The function failed due to a system error.

Comments

You can restore all session parameters to their default values by calling the Reset System (21) function.

The following tables list the session parameters that you can set (grouped by type), a description of each, and the functions each session parameter or group of session parameters affect.

For types that include more than one session parameter, you can set only one parameter at a time. For example, for the Translate Attributes type, you can set either ATTRB or NOATTRB. The default session parameters appear in bold.

Type	Parameter	Description	Affected Functions
Translate Attributes	**NOATTRB**	Translates bytes that are less than hexadecimal 1F to spaces (hexadecimal 20)	Copy Presentation Space (5) Copy Presentation Space to String (8) Copy Field to String (34)
	ATTRB	Passes all bytes as their original values	
Connect	**CONLOG**	Connects physically to the presentation space The session window associated with the presentation space becomes the active window.	Connect Presentation Space (1)
	CONPHYS	Connects logically to the presentation space The session window associated with the presentation space does not become the active window.	
Disconnect	**DISCONPHYS**	Disconnects the application from the session, disconnects the session from the host, and closes the session	Disconnect Presentation Space (2)
	DISCONLOG	Disconnects the application from the session, but does not disconnect from the host or close the session	
Mode	**PSMODE**	Writes data to the screen	
	RAWMODE	Simulates emulator transmits or host data	Copy Presentation Space to String (8) Copy String to Presentation Space (15)

Type	Parameter	Description	Affected Functions
Extended Attributes	**NOEAB**	Copies data without extended attributes	Copy Presentation Space (5) Copy Presentation Space to String (8) Copy String to Presentation Space (15) Copy String to Field (33) Copy Field to String (34)
	EAB	Copies both the text and the extended attributes Each character in the returned string is followed immediately by the attribute information for that character. For T 27, UTS 20, and UTS 40 sessions, the DataString must be twice the size of the presentation space (one character and one character attribute byte for each presentation space position). For UTS 60 sessions, the DataString must be three times the size of the presentation space (one character, one character attribute byte, and one color attribute byte for each presentation space position). For information about the attribute bytes returned by T 27 and UTS, refer to Appendix B, "Attribute Values."	
Strings	**STRLEN**	Requires an explicit length for all strings	Search Presentation Space (6) Copy String to Presentation Space (15) Search Field (30) Copy String to Field (33) Copy Field to String (34)
	STREOT	Requires an end-of-text character at the end of all strings rather than an explicit length	

Type	Parameter	Description	Affected Functions
End-of-Text	EOT=c	Defines the end-of-text (EOT) character for the end of a string The EOT character must be a one-byte literal character. Do not insert a space between the equal sign and the EOT character. The EOT character is used only if the STREOT session parameter is set. The default EOT character is a binary zero.	Send Key (3) Copy Presentation Space (5) Search Presentation Space (6) Copy Presentation Space to String (8) Copy String to Presentation Space (15) Search Field (30) Copy String to Field (33) Copy Field to String (34)
Escape	ESC=c	Defines the escape character used in the Send Key (3) and Get Key (51) functions The escape character can be any ASCII character. Do not insert any spaces between the equal sign and the escape character. The default escape character is @.	Send Key (3) Get Key (51)
Window Handle	**NOHWND104**	Omits the window handle in the returned buffer	Window Status (104)
	HWND104	Includes the window handle in the returned buffer	
Pause	**FPAUSE**	Pauses for the length of time specified by the Pause (18) function	Pause (18) Start Host Notification (23)
	IPAUSE	Once a Start Host Notification (23) call is made, pauses until a host event occurs	

Type	Parameter	Description	Affected Functions
Search	**SRCHALL**	Searches the entire presentation space or the entire field, depending on which function is called	Search Presentation Space (6) Search Field (30)
	SRCHFROM	Searches from the specified presentation space position to the end of the presentation space or the end of the field, depending on which function is called	
Wait	**TWAIT**	Waits for up to one minute before returning a parameter that indicates whether the session is ready to receive input If the session becomes ready for input within that time, the function returns WHLLOK (0) as soon as the session is ready. If the session does not become ready for input within that time, the function returns WHLLINHIBITED (5) at the end of the minute.	Wait (4)
	NWAIT	Immediately returns the current status of the session (that is, whether it can accept input)	
AutoOpen	**AUTOOPEN**	AUTOOPEN opens a session when a shortname points to a session that isn't running.	Affects most functions that accept a shortname input parameter, including: Query Sessions (10) Query Session Status (22) Start Host Notification (23) Query Host Update (24) Start Keystroke Intercept (50) Get Key (51) Post Intercept Status (52) Stop Keystroke Intercept (53) Convert Position or RowCol (99) Window Status (104)
	NOAUTOOPEN	NOAUTOOPEN will not open a session when a shortname points to a session that is not running.	

See Also

Connect Presentation Space (1), Disconnect Presentation Space (2), Send Key (3), Wait (4), Copy Presentation Space (5), Search Presentation Space (6), Copy Presentation Space to String (8), Copy String to Presentation Space (15), Pause (18), Reset System (21), Start Host Notification (23), Search Field (30), Copy String to Field (33), Copy Field to String (34), Window Status (104)

Query Sessions (10)

The Query Sessions function returns the number of all currently open sessions. For each session, this function also returns a 12-byte data string that contains the following information:

- Short name of the session (if any)
- File name of the session
- Type of session (T 27, UTS)
- Size of the presentation space

Prerequisites

None.

Syntax

```
WinHLLAPI(FunctionNumber,DataString,DataLength,
   PSPosition_ReturnCode)
```

Call Parameters

Parameter	Value
FunctionNumber	QUERYSESSIONS (10)
DataString	A data string used to return the session information. The string must be long enough to accommodate 12 bytes for each session.
DataLength	The length of the *DataString*.
PSPosition_ReturnCode	Not applicable.

Return Parameters

Parameter	Value	Description
DataString	A data string.	See “Comments.”
DataLength	The number of currently open sessions.	Not applicable.
PSPosition_ReturnCode	WHLLOK (0)	Function succeeded.
	WHLLPARAMETERERROR (2)	The *DataLength* is not a multiple of 12.
	WHLLSYSERROR (9)	The function failed due to a system error.

Comments

The returned *DataString* for each session is formatted as shown in the following table:

Byte	Description
1	The short name (null if the session has no short name)
2–9	The file name of the session (up to 8 characters)
10	The session type (A for T 27, O for UTS)
11–12	A binary number that indicates the presentation space size

The return value of *DataLength* is set only when the *PSPosition_ReturnCode* is WHLLOK (0) or WHLLPARAMETERERROR (2).

See Also

Connect Presentation Space (1), Query Session Status (22), Associate Profile (911), Remove Profile (912), Get Associations 913), Find File Name (914), Find Short Name(915)

Query Sessions Full (910)

The Query Sessions Full (910) function is the same as Query Sessions (10) except that it additionally provides the full path of the session file (using characters 13 through 272 of each entry) in its returned values.

Prerequisites

None.

Syntax

`WinHLLAPI(`*`FunctionNumber`*`,`*`DataString`*`,`*`DataLength`*`,`
`PSPosition_`*`ReturnCode`*`)`

Call Parameters

Parameter	Value
FunctionNumber	QUERYSESSIONS (10)
DataString	A data string used to return the session information. The string must be long enough to accommodate 272 bytes for each session.
DataLength	The length of the *DataString*.
PSPosition_ReturnCode	Not applicable.

Return Parameters

Parameter	Value	Description
DataString	A data string.	See "Comments."
DataLength	The number of currently open sessions.	Not applicable.
PSPosition_ReturnCode	WHLLOK (0)	Function succeeded.
	WHLLPARAMETERERROR (2)	The *DataLength* is not a multiple of 272.
	WHLLSYSERROR (9)	The function failed due to a system error.

Comments

The returned *DataString* for each session is formatted as shown in the following table:

Byte	Description
2–9	The short filename of the session (up to 8 characters)
10	The session type (A for T 27, O for UTS)
11–12	A binary number that indicates the presentation space size
13-272	The full path of the session file

See Also

Query Sessions (10).

Copy OIA (13)

The Copy OIA function returns the information about the operator information area (status line) and other information (such as the cursor shape). The information returned varies from one emulator to another.

Prerequisites

Connect Presentation Space (1)

Syntax

```
WinHLLAPI(FunctionNumber,DataString,DataLength,
   PSPosition_ReturnCode)
```

Call Parameters

Parameter	Value
FunctionNumber	COPYOIA (13)
DataString	A data string for the OIA information that will be returned by this function. For UTS, this string must be at least 103 bytes. For T 27, the string length varies, depending on whether you want this function to copy any T 27 user messages (that is, the row below the last defined row). If you do not want to copy these messages, the string must be 96 bytes. If you do want to copy these messages, this string must be at least 120 bytes.
DataLength	The number of bytes in the *DataString*. This parameter must be at least 120 to ensure that the *DataString* is large enough to hold all of the OIA information.
PSPosition_ReturnCode	Not applicable.

Return Parameters

Parameter	Value	Description
DataString	A 96- to 120-byte string containing OIA information.	See "Comments."
DataLength	Not applicable.	Not applicable.

Parameter	Value	Description
PSPosition_Return Code	WHLLOK (0)	Function succeeded.
	WHLLNOTCONNECTED (1)	Your application is not connected to a session.
	WHLLPARAMETERERROR (2)	The *DataLength* contains an invalid value; the status information was not copied.
	WHLLINHIBITED (5)	The status information copied successfully, but the session is locked (input inhibited).
	WHLLSYSERROR (9)	The function failed due to a system error.

Comments

The string returned in the *DataString* contains the following information:

Byte	Description
1	Emulator type
2–83	Reserved for future use
84–85	Information common to all emulators
86–120	Information specific to each emulator

The following tables display the OIA data for each emulator:

T 27 OIA Data

Byte	Description	Value
1	Emulator type	3
2–81	Reserved	Spaces
82–83	Reserved	0
84	Connection status	0 = Normal 1 = Broken

T 27 OIA Data, continued

Byte	Description	Value
85	Cursor shape	0 = Block 1 = Underline 2 = Vertical bar
86	Current page number	1-256
87	Total number of pages	1-256
88	Editing mode	0 = Overtype 1 = Insert in line 2 = Insert in page
89	Forms mode	0 = off 1 = on
90	Transmit mode	0 = off 1 = on
91	Control mode	0 = off 1 = on
92	Receive mode	0 = off 1 = on
93	LTAI	0 = off 1 = on
94	Enquire mode	0 = off 1 = on
95	Search mode	0 = off 1 = on
96	Transmit line mode	0 = off 1 = on
97	Connection status (QuickApp® only)	0 = Normal 0x80 = Broken
98–103	Reserved	0
104–119	User message*	ASCII (padded with blanks)
120	Reserved for ASCII 0	0x00

* The T 27 user message area is the first line below the regular presentation space and is used for messages sent by the host. Character attributes (such as bright, underline) are supported on this line, but they are not copied using this function. To retrieve the character attributes, use the Copy Presentation Space To String (8) function, with the startup position set to the first character after the normal presentation space ends.

Note: This function only allows you to capture the first 15 characters of the user message (which starts on line 25).

UTS OIA Data

Byte	Description	Value
1	Emulator type	2
2–81	Reserved	Spaces
82	Reserved	0
83	Terminal model	0 = UTS 20 1 = UTS 40 2 = UTS 60 3 = UTS 50
84	Connection status	0 = No Poll/ The connection is not active. 1 = Poll/ This is the normal case, everything is fully up. 2 = Connection Suspended/ The connection is broken. (Connection Suspended is available only with the PEPGate Client transport)
85	Cursor shape	0 = Block 1 = Underline 2 = Vertical bar
86	Current page number	1–256
87	Total number of pages	1–256
88	Editing mode	0 = Overtype 1 = Insert
89	Transmit mode	0 = Transmit ALL mode 1 = Transmit VAR mode 2 = Transmit CHAN mode
90	Keyboard lock (wait indicator)	0 = Keyboard unlocked 1 = Keyboard locked

91	Screen color	0xBF*, where
		B = Background color (0–7)
		F = Foreground color (0–7)

92	Status line color	0xBF*, where
		B = Background color (0–7)
		F = Foreground color (0–7)

UTS OIA Data, continued

Byte	Description	Value

93	Control Page protected fields color	0xBF*, where
		B = Background color (0–7)
		F = Foreground color (0–7)

94	Control Page unprotected fields color	0xBF*, where
		B = Background color (0–7)
		F = Foreground color (0–7)

*The color value is the color specified by the host, not the color specified by the emulator: 0 = Black, 1 = Red, 2 = Green, 3 = Yellow, 4 = Blue, 5 = Magenta, 6 = Cyan, and 7 = White.		
95	Current page type	0 = Workstation page 1 = System page 2 = Control Page
96	Control Page type	0 = Normal Control Page 1 = Extended Control Page
97	Connection status (QuickApp only)	0 = Normal 0x80 = Broken
98	Print mode	0 = Print PRNT Mode 1 = Print FORM Mode 2 = Print XPAR Mode
99	System message	0 = No System Message 1 = System Message
100	FCC transmit type	0 = No FCCs 1 = Expanded FCCs (UTS 20 and UTS 40) 2 = Color FCCs (UTS 60)
101	Emphasis transmit type	0 = No emphasis 1 = E2 2 = E3
102	Print DID value	0x20–0x7F
103	Read DID value	0x20–0x7F
104	Message waiting	0 = No message waiting 1 = Message waiting

UTS OIA Data, continued

Byte	Description	Value
105	Host changed Control Page indicator	1= Host accessed the Control Page (via an "Esc o" sequence) since the last time a transmit was done.
		0= UTS Control Page not accessed since the last time a transmit was done.
106	Host changed Screen indicator	1= Host placed at least 1 character on the 24x80 screen since the last transmit was done.
		0= Host placed no characters on the 24x80 screen since the last transmit was done.

See Also

Connect Presentation Space (1), Copy Presentation Space (5), Copy Presentation Space to String (8), Copy Field to String (34)

Query Field Attribute (14)

The Query Field Attribute function returns the attributes of the specified field in the presentation space. The information returned varies from one emulator to another.

Prerequisites

Connect Presentation Space (1)

Syntax

```
WinHLLAPI(FunctionNumber,DataString,DataLength,
   PSPosition_ReturnCode)
```

Call Parameters

Parameter	Value
FunctionNumber	QUERYFIELDATTRIBUTE (14)
DataString	Not applicable.
DataLength	Not applicable.
PSPosition_ReturnCode	Any presentation space position in the desired field.

Return Parameters

Parameter	Value	Description
DataString	Not applicable.	Not applicable.
DataLength	The value of the attribute byte for the specified field If the presentation space is unformatted, this value is zero.	Refer to the tables on the following pages for information about the returned values.
PSPosition_ReturnCode	WHLLOK (0)	Function succeeded.
	WHLLNOTCONNECTED (1)	Your application is not connected to a session.
	WHLLPOSITIONERROR (7)	An invalid presentation space position was passed.
	WHLLSYSERROR (9)	The function failed due to a system error.
	WHLLNOFIELD (24)	The field was not found, or the presentation space was unformatted.

Comments

The attributes for all the positions within a field should normally be the same. If a field has multiple attributes, the returned attribute value will be valid only for the specified presentation space position.

The information returned varies from one emulator to another, as shown in the following tables.

T 27 Field Attributes

Bit Position	Description	Value
7	Mode	0 = Nonforms mode 1 = Forms mode
6	Visibility	0 = Secure video 1 = Normal video
5	Unprotected/protected	0 = Unprotected 1 = Protected
4	Intensity	0 = Normal 1 = Bright
3	Justification	0 = Left justified 1 = Right justified
2	Protected types (valid when bit 5 is 1)	0 = Non-transmittable 1 = Protected
1	Reverse video	0 = Normal video 1 = Reverse video
0	Reserved	Space

Note: A non-transmittable field is a field that is not sent to the host when the user transmits. For example, labels on forms are typically non-transmittable fields. The information within field delimiters is normally transmitted to the host. These fields can be either protected or unprotected. If they are protected, the user cannot change them.

Since field delimiters themselves occupy a position in the presentation space, the field attribute for that position is non-transmittable.

UTS Field Attributes

Bit Position	Description	Value
7	Reserved	Space
6	Visibility	0 = Secure video 1 = Normal video
5	Unprotected/protected	0 = Unprotected 1 = Protected
4	Intensity	0 = Normal 1 = Dim
3	Justification	0 = Left justified 1 = Right justified
2	Field type	0 = Alphanumeric 1 = Numeric only
1	Reverse video	0 = Normal video 1 = Reverse video
0	FCC changed flag	0 = FCC has not been modified 1 = FCC has been modified

See Also Connect Presentation Space (1), Copy String to Field (33)

Copy String to Presentation Space (15)

The Copy String to Presentation Space function copies an ASCII string from your application to a specific location in the presentation space.

Prerequisites

Connect Presentation Space (1), Set Session Parameters (9)

Syntax

```
WinHLLAPI(FunctionNumber,DataString,DataLength,
   PSPosition_ReturnCode)
```

Call Parameters

Parameter	Value
FunctionNumber	COPYSTRTOPS (15)
DataString	The data string to copy to the presentation space. Make sure that the string is not larger than the presentation space.
DataLength	If the STRLEN session parameter is set (which is the default state), you must explicitly provide the length of the *DataString*. If the STREOT session parameter is set, this parameter is ignored.
PSPosition_ReturnCode	The presentation space position where you want to place the copied data.

Return Parameters

Parameter	Value	Description
DataString	Not applicable.	Not applicable.
DataLength	Not applicable.	Not applicable.

Parameter	Value	Description
PSPosition_ReturnCode	WHLLOK (0)	Function succeeded.
	WHLLNOTCONNECTED (1)	Your application is not connected to a session.
	WHLLPARAMETERERROR (2)	A *DataLength* of zero was specified.
	WHLLINHIBITED (5)	The presentation space is protected or inhibited, or inappropriate data (such as a field attribute byte) was passed. None of the string was copied.
	WHLLTRUNCATED (6)	The string was copied, but data was truncated at the end of the screen.
	WHLLPOSITIONERROR (7)	An invalid presentation space position was specified.
	WHLLSYSERROR (9)	The function failed due to a system error.

Comments

If you are copying a string to a formatted presentation space, use the Query Field Attribute (14) function to determine whether the presentation space position occurs in a protected field before copying the string to that location.

See Also

Connect Presentation Space (1), Set Session Parameters (9), Query Field Attribute (14), Copy String to Field (33)

Pause (18)

The Pause function causes your application to wait a specific amount of time or for an event to occur. Use this function instead of a timing loop.

Prerequisites

None.

Syntax

```
WinHLLAPI(FunctionNumber,DataString,DataLength,
   PSPosition_ReturnCode)
```

Call Parameters

Parameter	Value
FunctionNumber	PAUSE (18)
DataString	Not applicable.
DataLength	The number of half-seconds that the application should pause. For example, to pause 60 seconds, the *DataLength* should be 120.
PSPosition_ReturnCode	Not applicable.

Return Parameters

Parameter	Value	Description
DataString	Not applicable.	Not applicable.
DataLength	Not applicable.	Not applicable.
PSPosition_ReturnCode	WHLLOK (0)	The pause duration expired.
	WHLLSYSERROR (9)	The function failed due to a system error.
	WHLLPSCHANGED (26)	The presentation space has been updated.

Comments

If the FPAUSE session parameter is set (which is the default state), the application pauses until the amount of time specified by the *DataLength* expires.

If the IPAUSE session parameter is set and Start Host Notification (23) has been called, the application pauses the amount of time specified by the *DataLength* or until the host sends data to the session, whichever comes first.

See Also

Set Session Parameters (9), Start Host Notification (23), Query Host Update (24)

Query System (20)

The Query System function provides information about the WinHLLAPI and IHLLAPI DLL version numbers and level numbers.

Prerequisites

None.

Syntax

```
WinHLLAPI(FunctionNumber,DataString,DataLength,
   PSPosition_ReturnCode)
```

Call Parameters

Parameter	Value
FunctionNumber	QUERYSYSTEM (20)
DataString	A 35-byte buffer for incoming system data.
DataLength	Not applicable.
PSPosition_ReturnCode	Not applicable.

Return Parameters

Parameter	Value	Description
DataString	A 35-byte data string.	See “Comments.”
DataLength	Not applicable.	Not applicable.
PSPosition_ReturnCode	WHLLOK	Function succeeded.
	WHLLSYSERROR	The function failed due to a system error.

Comments

The returned *DataString* contains the following information:

Byte	Description
1	The Microsoft WinHLLAPI version number
2–3	The Microsoft WinHLLAPI level number
4–9	The Microsoft WinHLLAPI release date in month/day/year format (for example, 031996)
10–12	Reserved
13	U
14	E
15–16	The IHLLAPI version number
17–18	The IHLLAPI level number
19–35	Reserved

See Also

Query Sessions (10), Query Session Status (22)

Reset System (21)

The Reset System function sets all session parameters to their default values. This function also stops host event notification and disconnects any connected presentation spaces.

Prerequisites

None.

Syntax

```
WinHLLAPI(FunctionNumber,DataString,DataLength,
   PSPosition_ReturnCode)
```

Call Parameters

Parameter	Value
FunctionNumber	RESETSYSTEM (21)
DataString	Not applicable.
DataLength	Not applicable.
PSPosition_ReturnCode	Not applicable.

Return Parameters

Parameter	Value	Description
DataString	Not applicable.	Not applicable.
DataLength	Not applicable.	Not applicable.
PSPosition_ReturnCode	WHLLOK (0)	Function succeeded.
	WHLLSYSERROR (9)	The function failed due to a system error.

Comments

Your application should call this function before closing to ensure that other applications start up in a known environment.

See Also

Disconnect Presentation Space (2), Set Session Parameters (9), Start Host Notification (23), Stop Host Notification (25)

Query Session Status (22)

The Query Session Status function returns an 18-byte data string with the following information about a specified session:

- Short name
- File name
- Session type
- Number of rows and columns in the presentation space

Prerequisites

None.

Syntax

```
WinHLLAPI(FunctionNumber,DataString,DataLength,
   PSPosition_ReturnCode)
```

Call Parameters

Parameter	Value
FunctionNumber	QUERYSESSIONSTATUS (22)
DataString	An 18-byte data string containing the short name of the session plus 17 extra bytes (to reserve space for the returned data). The short name can be a space or null character if you want to retrieve data about the current presentation space.
DataLength	The number of bytes in the *DataString* (at least 18).
PSPosition_ReturnCode	Not applicable.

Return Parameters

Parameter	Value	Description
DataString	An 18-byte string.	See “Comments.”
DataLength	Not applicable.	Not applicable.

Parameter	Value	Description
PSPosition_ReturnCode	WHLLOK (0)	Function succeeded.
	WHLLNOTCONNECTED (1)	An invalid short name was specified.
	WHLLPARAMETERERROR (2)	An invalid *DataLength* was passed.
	WHLLSYSERROR (9)	The function failed due to a system error.

Comments

The returned *DataString* contains the following information:

Byte	Description
1	Short name of the session
2–9	File name of the session (up to eight characters)
10	Session type (A for T 27, O for UTS)
11	Reserved
12–13	Number of rows in the presentation space (binary number)
14–15	Number of columns in the presentation space (binary number)
16–18	Reserved

For an unopened session, this call returns only the short name and file name for the session. Bytes 10 through 18 in the returned *DataString* will contain null characters.

Use Query Sessions (10) to determine which sessions are open.

See Also

Query Sessions (10)

Start Host Notification (23)

This function enables notifying your application of changes in the presentation space or operator information area (OIA).

Prerequisites

Connect Presentation Space (1)

Syntax

```
WinHLLAPI(FunctionNumber,DataString,DataLength,
   PSPosition_ReturnCode)
```

Call Parameters

Parameter	Value
FunctionNumber	STARTHOSTNOTIFICATION (23)
DataString	A 7-byte string. See “Comments” for details.
DataLength	Not applicable.
PSPosition_ReturnCode	Not applicable.

Return Parameters

Parameter	Value	Description
DataString	Same as the *DataString* on the call parameter.	See “Comments.”
DataLength	Not applicable.	Not applicable.
PSPosition_ReturnCode	WHLLOK (0)	Function succeeded.
	WHLLNOTCONNECTED (1)	Your application is not connected to a session.
	WHLLPARAMETERERROR (2)	One or more parameters are invalid.
	WHLLSYSERROR (9)	The function failed due to a system error.

Comments

The DataString in the call parameter is a 7-byte string that contains the following information:

Byte	Description
1	Short name of the desired session, or a space or null character for the current session
2	P for presentation space update only, O for OIA update only, B for both presentation space and OIA updates
3–6	Not used; no error occurs if old IHLLAPI applications use these positions
7	Reserved

Once host notification is enabled, it remains enabled until your application calls Stop Host Notification (25) or Reset System (21).

If the IPAUSE session parameter is set and Start Host Notification (23) has been called, you can use the Pause (18) function to pause your application the amount of time specified by the Pause (18) function's DataLength or until the host updates the presentation space and/or OIA, whichever comes first.

See Also

Set Session Parameters (9), Pause (18), Reset System (21), Query Host Update (24), Stop Host Notification (25)

Query Host Update (24)

This function determines if the presentation space or operator information area (OIA) has been updated since Start Host Notification (23) was called or since this function was previously called.

Prerequisites

Start Host Notification (23)

Syntax

```
WinHLLAPI(FunctionNumber,DataString,DataLength,
   PSPosition_ReturnCode)
```

Call Parameters

Parameter	Value
FunctionNumber	QUERYHOSTUPDATE (24)
DataString	The short name of the desired session, or a space or null character for the current session.
DataLength	Not applicable; a length of 1 byte is implied.
PSPosition_ReturnCode	Not applicable.

Return Parameters

Parameter	Value	Description
DataString	Not applicable.	Not applicable.
DataLength	Not applicable.	Not applicable.

Parameter	Value	Description
PSPosition_ReturnCode	WHLLOK (0)	No updates occurred.
	WHLLNOTCONNECTED (1)	The specified session is invalid.
	WHLLNOTAVAILABLE (8)	Start Host Notification (23) was not called for the specified session prior to this function.
	WHLLSYSERROR (9)	The function failed due to a system error.
	WHLLOIAUPDATE (21)	One or more updates to the OIA occurred.
	WHLLPSUPDATE (22)	One or more updates to the presentation space occurred.
	WHLLBOTHUPDATE (23)	One or more updates to both the OIA and the presentation space occurred.

Comments

In this version of IHLLAPI, the OIA is not handled separately from the presentation space. This function returns WHLLBOTHUPDATE (23) for any update.

See Also

Start Host Notification (23), Stop Host Notification (25)

Stop Host Notification (25)

This function disables notifying your application of changes in the presentation space and/or operator information area (OIA).

Prerequisites

Start Host Notification (23)

Syntax

```
WinHLLAPI(FunctionNumber,DataString,DataLength,
   PSPosition_ReturnCode)
```

Call Parameters

Parameter	Value
FunctionNumber	STOPHOSTNOTIFICATION (25)
DataString	The short name of the desired session, or a space or null character for the current session.
DataLength	Not applicable; a length of 1 byte is implied.
PSPosition_ReturnCode	Not applicable.

Return Parameters

Parameter	Value	Description
DataString	Not applicable.	Not applicable.
DataLength	Not applicable.	Not applicable.
PSPosition_ReturnCode	WHLLOK (0)	Function succeeded.
	WHLLNOTCONNECTED (1)	Your application is not connected to a session.
	WHLLNOTAVAILABLE (8)	Start Host Notification (23) was not called for the specified session prior to this function.
	WHLLSYSERROR (9)	The function failed due to a system error.

Comments

Once host notification has been disabled, Query Host Update (24) can no longer determine whether the presentation space or OIA have been updated, and host events will not cause the Pause (18) function to return a return code to the application.

The Reset System (21) function also stops host notification.

See Also

Pause (18), Reset System (21), Start Host Notification (23), Query Host Update (24)

Search Field (30)

The Search Field function searches the specified field for a specified string.

Prerequisites

Connect Presentation Space (1)

Syntax

```
WinHLLAPI(FunctionNumber,DataString,DataLength,
   PSPosition_ReturnCode)
```

Call Parameters

Parameter	Value
FunctionNumber	SEARCHFIELD (30)
DataString	The string that you want this function to search for. If the STREOT session parameter is set, the last character in this string must be an end-of-text character.
DataLength	If the STRLEN session parameter is set (which is the default state), this is the length of the *DataString*. If the STREOT session parameter is set, this parameter is ignored.
PSPosition_ReturnCode	A position within the field that you want to search. If the SRCHALL session parameter is set (which is the default state), this function searches the entire field. If the SRCHFROM session parameter is set, this function begins searching at the specified location and stops at the end of the field.

Return Parameters

Parameter	Value	Description
DataString	Not applicable.	Not applicable.
DataLength	0 or greater than 0	If the value is zero, the string was not found. A value greater than zero indicates the presentation space position where the string begins.
PSPosition_ReturnCode	WHLLOK (0)	The function succeeded. Check the *DataLength* for the search result.
	WHLLNOTCONNECTED (1)	Your application is not connected to a session.
	WHLLPARAMETERERROR (2)	An invalid parameter was passed.
	WHLLPOSITIONERROR (7)	The *PSPosition_ ReturnCode* was either zero or greater than the presentation space size.
	WHLLSYSERROR (9)	The function failed due to a system error.
	WHLLNOFIELD (24)	The presentation space was unformatted.

Comments

This function applies only to formatted presentation spaces. For unformatted presentation spaces, use the Search Presentation Space (6) function.

See Also

Connect Presentation Space (1), Search Presentation Space (6), Set Session Parameters (9)

Find Field Position (31)

The Find Field Position function returns the beginning presentation space position of a specified field.

Prerequisites

Connect Presentation Space (1)

Syntax

```
WinHLLAPI(FunctionNumber,DataString,DataLength,
   PSPosition_ReturnCode)
```

Call Parameters

Parameter	Value
FunctionNumber	FINDFIELDPOSITION (31)
DataString	A two-character code that specifies which field you want to find the beginning position of. See “Comments” for details.
DataLength	Not applicable; a length of 2 bytes is implied.
PSPosition_ReturnCode	Any presentation space position within the field where you want to begin the search.

Return Parameters

Parameter	Value	Description
DataString	Not applicable.	Not applicable.
DataLength	The starting position of the field.	A zero for this parameter indicates that the field was not found, the presentation space was unformatted, or the field length was zero.

Parameter	Value	Description
PSPosition_ReturnCode	WHLLOK (0)	Function succeeded.
	WHLLNOTCONNECTED (1)	Your application is not connected to a session.
	WHLLPARAMETERERROR (2)	An incorrect parameter was specified.
	WHLLPOSITIONERROR (7)	An invalid presentation space position was specified.
	WHLLSYSERROR (9)	The function failed due to a system error.
	WHLLNOFIELD (24)	The field was not found, or the presentation space was unformatted.
	WHLLZEROLENFIELD (28)	The field length is zero.

Comments

The *DataString* in the call parameter is a two-character code that specifies which field you want to find the beginning position of. These codes must be in uppercase.

Code	Field
<space><space>	The field specified by the *PSPosition_ReturnCode*
T<space>	The field specified by the *PSPosition_ReturnCode*
P<space>	Previous protected or unprotected field
PP	Previous protected field
PU	Previous unprotected field
N<space>	Next protected or unprotected field
NP	Next protected field
NU	Next unprotected field

This function applies only to formatted presentation spaces.

See Also

Connect Presentation Space (1), Find Field Length (32)

Find Field Length (32)

The Find Field Length function returns the length of a specified field.

Prerequisites

Connect Presentation Space (1)

Syntax

```
WinHLLAPI(FunctionNumber,DataString,DataLength,
   PSPosition_ReturnCode)
```

Call Parameters

Parameter	Value
FunctionNumber	FINDFIELDLENGTH (32)
DataString	A two-character code that specifies which field you want the length of. See “Comments” for details.
DataLength	Not applicable; a length of 2 bytes is implied.
PSPosition_ReturnCode	Any presentation space position within the field whose length you want to find.

Return Parameters

Parameter	Value	Description
DataString	Not applicable.	Not applicable.
DataLength	The length of the specified field.	A zero indicates that the field was not found, the presentation space was unformatted, or the field length was zero.

Parameter	Value	Description
PSPosition_ReturnCode	WHLLOK (0)	Function succeeded.
	WHLLNOTCONNECTED (1)	Your application is not connected to a session.
	WHLLPARAMETERERROR (2)	An incorrect parameter was specified.
	WHLLPOSITIONERROR (7)	An invalid presentation space position was specified.
	WHLLSYSERROR (9)	The function failed due to a system error.
	WHLLNOFIELD (24)	The field was not found, or the presentation space was unformatted.
	WHLLZEROLENFIELD (28)	The field length is zero.

Comments

The *DataString* in the call parameter is a two-character code that specifies which field you want the length of. These codes must be in uppercase.

Value	Field
<space><space>	The field specified by the *PSPosition_ReturnCode*
T<space>	The field specified by the *PSPosition_ReturnCode*
P<space>	Previous protected or unprotected field
PP	Previous protected field
PU	Previous unprotected field
N<space>	Next protected or unprotected field
NP	Next protected field
NU	Next unprotected field

This function applies only to formatted presentation spaces.

See Also

Connect Presentation Space (1), Find Field Position (31)

Copy String to Field (33)

The Copy String to Field function copies a string of characters from your application to a specified field.

Prerequisites

Connect Presentation Space (1)

Syntax

```
WinHLLAPI(FunctionNumber,DataString,DataLength,
   PSPosition_ReturnCode)
```

Call Parameters

Parameter	Value
FunctionNumber	COPYSTRINGTOFIELD (33)
DataString	The data string to copy to the specified field. If the STREOT session parameter is set, this string must contain an end-of-text character.
DataLength	If the STRLEN session parameter is set (which is the default state), you must explicitly provide the length of the DataString. If the STREOT session parameter is set, this parameter is ignored.
PSPosition_ReturnCode	Any presentation space position within the field where you want to place the data string. The data string is always placed at the first position of the field. If the field is right-justified, the emulator moves the string to the appropriate position.

Return Parameters

Parameter	Value	Description
DataString	Not applicable.	Not applicable.
DataLength	Not applicable.	Not applicable.

Parameter	Value	Description
PSPosition_ReturnCode	WHLLOK (0)	Function succeeded.
	WHLLNOTCONNECTED (1)	Your application is not connected to a session.
	WHLLPARAMETERERROR (2)	A *DataLength* of zero was specified.
	WHLLINHIBITED (5)	The field is protected or inhibited, or inappropriate data (such as a field attribute byte) was passed. None of the string was copied.
	WHLLTRUNCATED (6)	The string was copied, but data was truncated.
	WHLLPOSITIONERROR (7)	An invalid presentation space position was specified.
	WHLLSYSERROR (9)	The function failed due to a system error.
	WHLLNOFIELD (24)	The presentation space is unformatted.
	WHLLZEROLENFIELD (28)	The specified field has a length of zero.

Comments

This function applies only to formatted presentation spaces.

In addition, this function stops copying the string when it encounters one of the following:

- The end of the field
- The number of characters specified by the DataLength (if the STRLEN session parameter is set)
- An end-of-text character (if the STREOT session parameter is set)

See Also

Connect Presentation Space (1), Set Session Parameters (9), Copy String to Presentation Space (15)

Copy Field to String (34)

The Copy Field to String function copies all the characters from a specified field to a data string in your application.

Prerequisites

Connect Presentation Space (1)

Syntax

```
WinHLLAPI(FunctionNumber,DataString,DataLength,
   PSPosition_ReturnCode)
```

Call Parameters

Parameter	Value
FunctionNumber	COPYFIELDTOSTRING (34)
DataString	An empty data string large enough to hold all the data copied from the field. If the EAB session parameter is set, this string must be two or three times the size of the presentation space (see “Comments”).
DataLength	If the STRLEN session parameter is set (which is the default state), you must explicitly provide the length of the *DataString*. If the STREOT session parameter is set, this parameter is ignored.
PSPosition_ReturnCode	Any presentation space position within the field that you want to copy. The copy always begins at the first position of the field. If the field is right-justified, the emulator moves the string to the appropriate position.

Return Parameters

Parameter	Value	Description
DataString	A string containing the contents of the field.	See “Comments.”
DataLength	Not applicable.	Not applicable.

Parameter	Value	Description
PSPosition_ReturnCode	WHLLOK (0)	Function succeeded.
	WHLLNOTCONNECTED (1)	Your application is not connected to a session.
	WHLLPARAMETERERROR (2)	An invalid parameter was passed.
	WHLLTRUNCATED (6)	The field in the presentation space and the string in your application are not the same size. Some data might have been truncated.
	WHLLPOSITIONERROR (7)	An invalid presentation space position was specified.
	WHLLSYSERROR (9)	The function failed due to a system error.
	WHLLNOFIELD (24)	The field was not found, or the presentation space was unformatted.
	WHLLZEROLENFIELD (28)	The specified field has a length of zero.

Comments

This function applies only to formatted presentation spaces.

If the NOEAB session parameter is set (which is the default state), this function copies only the data in the field; it does not copy any extended attributes.

If the EAB session parameter is set, each character in the returned string is followed immediately by the attribute information for that character. Therefore, you must allocate a *DataString* that is long enough to accommodate both the text and the attribute information.

For T 27, UTS 20, and UTS 40 sessions, the *DataString* must be twice the size of the field (one character and one character attribute byte for each field position). For example, if the field includes 10 presentation space positions, the *DataString* must be 20 bytes. For UTS 60 sessions, the *DataString* must be three times the size of the field (one character, one character attribute byte, and one color attribute byte for each field position).

For information about the attribute bytes returned by T 27 and UTS, refer to Appendix B, “Attribute Values.”

If the NOATTRB session parameter is set (which is the default state), bytes that are less than 0x1F are translated to spaces (0x20). If the ATTRB session parameter is set, all bytes are passed as their original values.

See Also

Connect Presentation Space (1), Copy Presentation Space (5), Copy Presentation Space to String (8), Set Session Parameters (9)

Set Cursor (40)

The Set Cursor function lets you position the cursor within the presentation space.

Prerequisites

Connect Presentation Space (1)

Syntax

```
WinHLLAPI(FunctionNumber,DataString,DataLength,
   PSPosition_ReturnCode)
```

Call Parameters

Parameter	Value
FunctionNumber	SETCURSOR (40)
DataString	Not applicable.
DataLength	Not applicable.
PSPosition_ReturnCode	The presentation space position where you want to position the cursor. The upper left corner (row 1, column 1) of the presentation space is position 1.

Return Parameters

Parameter	Value	Description
DataString	Not applicable.	Not applicable.
DataLength	Not applicable.	Not applicable.

Parameter	Value	Description
PSPosition_ReturnCode	WHLLOK (0)	Function succeeded.
	WHLLNOTCONNECTED (1)	Your application is not connected to a session.
	WHLLPSBUSY (4)	The presentation space is busy; the cursor could not be positioned.
	WHLLPOSITIONERROR (7)	You specified a presen-tation space position less than 1 or greater than the presentation space size.
	WHLLSYSERROR (9)	The function failed due to a system error.

Comments

Use the Query Cursor Location (7) function to retrieve the current position of the cursor.

See Also

Connect Presentation Space (1), Query Cursor Location (7)

Start Keystroke Intercept (50)

The Start Keystroke Intercept function lets your application intercept keystrokes typed in the session by the user.

Prerequisites

Connect Presentation Space (1)

Syntax

```
WinHLLAPI(FunctionNumber,DataString,DataLength,
   PSPosition_ReturnCode)
```

Call Parameters

Parameter	Value
FunctionNumber	STARTKSINTERCEPT (50)
DataString	A 6-byte string for each session. See “Comments” for details.
DataLength	Variable (256 bytes is recommended).
PSPosition_ReturnCode	Not applicable.

Return Parameters

Parameter	Value	Description
DataString	Not applicable.	Not applicable.
DataLength	Not applicable.	Not applicable.

Parameter	Value	Description
PSPosition_ReturnCode	WHLLOK (0)	Function succeeded.
	WHLLNOTCONNECTED (1)	Your application is not connected to a session.
	WHLLPARAMETERERROR (2)	One or more call parameters are invalid.
	WHLLPSBUSY (4)	The presentation space is busy.
	WHLLSYSERROR (9)	The function failed due to a system error.
	WHLLCANCEL (0xF002)	The asynchronous function was canceled.

Comments

The DataString in the call parameter is a 6-byte string for each session that contains the following information:

Byte	Description
1	Short name of the desired session, or a space or null character for the current session
2	D = intercept only AID keystrokes, L = intercept all keystrokes
3–6	Reserved

Once this function is called, the intercepted keystrokes can be handled in the following ways:

- Received with the Get Key (51) function
- Accepted or rejected with the Post Intercept Status (52) function
- Sent to the same session or another session with the Send Key (3) function
- Used in another manner appropriate for your application

If the second position of the *DataString* is D, only AID (Attention identification) keystrokes are intercepted. AID keystrokes are keystrokes that directly cause a change in the screen display, such as letters. Non-AID keystrokes, such as Enter or function keys, do not directly change the display.

See Also

Connect Presentation Space (1), Send Key (3), Get Key (51), Post Intercept Status (52), Stop Keystroke Intercept (53)

Get Key (51)

The Get Key function lets your application intercept keystrokes from a session for which keystroke intercept has been enabled.

Prerequisites

Start Keystroke Intercept (50)

Syntax

```
WinHLLAPI(FunctionNumber,DataString,DataLength,
   PSPosition_ReturnCode)
```

Call Parameters

Parameter	Value
FunctionNumber	GETKEY (51)
DataString	An 8-byte string. The first byte is the short name of the desired session, or a space or null character for the current session. Bytes 2–8 are reserved for return data.
DataLength	Not applicable; a length of 8 bytes is implied.
PSPosition_ReturnCode	Not applicable.

Return Parameters

Parameter	Value	Description
DataString	An 8-byte string.	See “Comments.”
DataLength	The number of characters in the returned *DataString*.	Not applicable.

Parameter	Value	Description
PSPosition_ReturnCode	WHLLOK (0)	Function succeeded.
	WHLLNOTCONNECTED (1)	Your application is not connected to a session.
	WHLLINHIBITED (5)	The Start Keystroke intercept (50) function was called with the D option, but no AID keystrokes were intercepted.
	WHLLNOTAVAILABLE (8)	The Start Keystroke Intercept (50) function was not called prior to this function.
	WHLLSYSERROR (9)	The function failed due to a system error.
	WHLLUNDEFINEDKEY (20)	The user typed an invalid keystroke combination.
	WHLLNOKEYSTROKES (25)	No keystrokes are available in the keystroke queue.
	WHLLKEYOVERFLOW (31)	The keystroke queue has overflowed and keystrokes were lost.

Comments

The returned *DataString* is an 8-byte string that contains the following information:

Byte	Description
1	Short name of the desired session, or a space or null character for the current session
2	A = an ASCII character; S = a special modifier such as Shift, Ctrl, or Alt
3–8	Keystroke(s) Unused bytes are set to null characters. See “Comments” for details.

The keystrokes typed by the user are queued by the IHLAPI32.DLL. This function reads the keystrokes from the queue one at a time.

AID keystrokes are keystrokes that directly cause a change in the screen display, such as letters. Non-AID keystrokes, such as Enter or function keys, do not directly change the display.

The ESC=c session parameter determines which character is used as the escape character. In the following examples, the default escape character (@) is used.

The following modifiers indicate when the Alt, Shift, or Ctrl keys were pressed in conjunction with another key:

Modifier	Meaning
@A	The Alt key was pressed.
@S	The Shift key was pressed.
@r	The Ctrl key was pressed.

Returned DataString Examples

The following table shows possible returned *DataStrings*:

DataString	Description
BAt	B is the short name of the session, A indicates that the following keystroke is ASCII, and t is the keystroke (bytes 4–8 are null).
CS@Aa	C is the short name of the session, S indicates that the following keystroke is a modifier, @A indicates that the modifier is the Alt keystroke, and a is the keystroke that was pressed in conjunction with the Alt key (bytes 6–8 are null).
DS@rA	D is the short name of the session, S indicates that the following keystroke is a modifier, @r indicates that the modifier is the Ctrl keystroke, and A is the keystroke that was pressed in conjunction with the Ctrl key (bytes 6–8 are null). The uppercase A indicates that the Shift key was also pressed.

See Also

Send Key (3), Set Session Parameters (9), Start Keystroke Intercept (50), Post Intercept Status (52), Stop Keystroke Intercept (53)

Post Intercept Status (52)

The Post Intercept Status function indicates whether to accept or reject a keystroke obtained with the Get Key (51) function.

Prerequisites

Start Keystroke Intercept (50)

Syntax

```
WinHLLAPI(FunctionNumber,DataString,DataLength,
   PSPosition_ReturnCode)
```

Call Parameters

Parameter	Value
FunctionNumber	POSTINTERCEPTSTATUS (52)
DataString	A 2-byte string. The first byte is the short name of the desired session, or a space or null character for the current session. The second byte is A (accept the keystroke) or R (reject the keystroke).
DataLength	Not applicable; a length of 2 bytes is implied.
PSPosition_ReturnCode	Not applicable.

Return Parameters

Parameter	Value	Description
DataString	Not applicable.	Not applicable.
DataLength	Not applicable.	Not applicable.
PSPosition_ReturnCode	WHLLOK (0)	Function succeeded.
	WHLLNOTCONNECTED (1)	Your application is not connected to a session.
	WHLLPARAMETERERROR (2)	One or more call parameters are invalid.
	WHLLNOTAVAILABLE (8)	The Start Key-stroke Intercept (50) function was not called prior to this function.
	WHLLSYSERROR (9)	The function failed due to a system error.

Comments

If a keystroke is rejected, a beep sounds, and the keystroke is not displayed on the screen or sent to the host.

See Also

Start Keystroke Intercept (50), Get Key (51), Stop Keystroke Intercept (53)

Stop Keystroke Intercept (53)

The Stop Keystroke Intercept function stops your application from being able to intercept keystrokes.

Prerequisites

Start Keystroke Intercept (50)

Syntax

```
WinHLLAPI(FunctionNumber,DataString,DataLength,
   PSPosition_ReturnCode)
```

Call Parameters

Parameter	Value
FunctionNumber	STOPKSINTERCEPT (53)
DataString	The short name of the desired session, or a space or null character for the current session.
DataLength	Not applicable; a length of 1 byte is implied.
PSPosition_ReturnCode	Not applicable.

Return Parameters

Parameter	Value	Description
DataString	Not applicable.	Not applicable.
DataLength	Not applicable.	Not applicable.
PSPosition_ReturnCode	WHLLOK (0)	Function succeeded.
	WHLLNOTCONNECTED (1)	Your application is not connected to a session.
	WHLLNOTAVAILABLE (8)	The Start Keystroke Intercept (50) function was not called prior to this function.
	WHLLSYSERROR (9)	The function failed due to a system error.

Comments

Once keystroke intercept has been disabled, the Get Key (51) function can no longer intercept keystrokes.

See Also

Start Keystroke Intercept (50), Get Key (51), Post Intercept Status (52)

Send File (90)

The SEND FILE function is used to transfer a file from the local workstation to the host session. The host session must have access to the IND$FILE 3270 file transfer product, or its equivalent. File transfers are subject to the same host state rules as user initiated file transfers. TSO must be at a READY prompt or at the ISPF TSO command screen, CMS must be at a CMS command line prompt and in VM READ or RUNNING state.

Prerequisites

Connect Presentation Space (1)

Syntax

```
WinHLLAPI(FunctionNumber,DataString,DataLength,
   PSPosition_ReturnCode)
```

Call Parameters

Parameter	Value
FunctionNumber	SENDFILE (90)
DataString	The data string must consist of between 1 and 128 bytes containing one of the following: For CMS: *localfilename* S:*hostfilename* (options) For TSO: *localfilename* S:*hostfilename* options where *localfilename* refers to the local path and file name of the file to be transferred and *hostfilename* refers to the host dataset name (including the member name, if needed) for MVS or the CMS file name for CMS to be created. S:, which is optional, refers to the presentation space ID of the presentation space to be used for the transfer. If not supplied, the currently connected presentation space is used.
DataLength	The length of *DataString*
PSPosition_ReturnCode	Not applicable.

Return Parameters

Parameter	Value	Description
PSPosition_ReturnCode	WHLLFTXCOMPLETE (3)	Function succeeded.
	WHLLNOTCONNECTED (1)	Your application is not connected to a session.

Parameter	Value	Description
PSPosition_ReturnCode	WHLLPARAMETERERROR (2)	Invalid call parameter.
	WHLLFTXABORTED (27)	File transfer failed.
	WHLLSYSERROR (9)	A system error occurred.

Comments None.

See Also Receive File (91)

Receive File (91)

The RECEIVE FILE function is used to transfer a file from the host session to the local workstation. The host session must have access to the IND$FILE 3270 file transfer product, or its equivalent. File transfers are subject to the same host state rules as user initiated file transfers. TSO must be at a READY prompt or at the ISPF TSO command screen, CMS must be at a CMS command line prompt and in VM READ or RUNNING state.

Prerequisites

Connect Presentation Space (1)

Syntax

```
WinHLLAPI(FunctionNumber,DataString,DataLength,
   PSPosition_ReturnCode)
```

Call Parameters

Parameter	Value
FunctionNumber	RECEIVEFILE (91)
DataString	The data string must consist of between 1 and 128 bytes containing one of the following: For CMS: *localfilename* S:*hostfilename* (options) For TSO: *localfilename* S:*hostfilename* options where *localfilename* refers to the local path and file name of the file to be created and *hostfilename* refers to the existing host dataset name (including the member name, if needed) for MVS or the CMS file name for CMS. S:, which is optional, refers to the presentation space ID of the presentation space to be used for the transfer. If not supplied, the currently connected presentation space is used.
DataLength	The length of *DataString*
PSPosition_ReturnCode	Not applicable.

Return Parameters

Parameter	Value	Description
PSPosition_ReturnCode	WHLLFTXCOMPLETE (3)	Function succeeded.
	WHLLNOTCONNECTED (1)	Your application is not connected to a session.

Parameter	Value	Description
PSPosition_ReturnCode	WHLLPARAMETERERROR (2)	Invalid call parameter.
	WHLLFTXABORTED (27)	File transfer failed.
	WHLLSYSERROR (9)	A system error occurred.

Comments None.

See Also Send File (90)

Convert Position or RowCol (99)

The Convert Position or RowCol function converts the presentation space position into row and column coordinates, or row and column coordinates into a presentation space position, depending on the call parameters passed by your application.

Prerequisites

None

Syntax

```
WinHLLAPI(FunctionNumber,DataString,DataLength,
   PSPosition_ReturnCode)
```

Call Parameters

Parameter	Value
FunctionNumber	CONVERT (99)
DataString	A two-byte string. The first byte is the short name of the desired session, or a space or null character for the current session. The second byte is P (convert the presentation space position into row and column coordinates) or R (convert row and column coordinates into a presentation space position).
DataLength	If you're converting a presentation space position into row and column coordinates, set this parameter to zero. If you're converting row and column coordinates into a presentation space position, this is the row number being converted.
PSPosition_ReturnCode	If you're converting a presentation space position into row and column coordinates, this parameter is the presentation space position being converted. If you're converting row and column coordinates into a presentation space position, this parameter is the column number being converted.

Return Parameters

Parameter	Value	Description
DataString	Not applicable.	Not applicable.
DataLength	If you're converting a presentation space position into row and column coordinates, this is the row number. If you're converting row and column coordinates into a presentation space position, this is not applicable.	If the function returns a zero for this parameter, you specified a presentation space position that was larger than the number of rows in the presentation space.
PSPosition_ReturnCode	WHLLOK = 0	An invalid presentation space position or column was specified.
	WHLLOK > 0	If you're converting a presentation space position into row and column coordinates, this is the column number. If you're converting row and column coordinates into a presentation space position, this is the presentation space position.
	WHLLINVALIDPSID (9998)	Your application specified an invalid short name or a system error occurred.
	WHLLINVALIDRC (9999)	The second character in the DataString was not an uppercase P or R.

Comments

When you specify a short name in the call parameters, a connection to that session is established automatically. You do not have to invoke the Connect Presentation Space (1) function before invoking the Convert Position or RowCol (99) function.

Since the Convert Position or RowCol (99) function does not return a standard return code, your application could obtain misleading information if you use a common error-handling routine for all of the IHLLAPI functions. You should develop a special error-handling routine for this function. (For more information, refer to Appendix A, “Return Codes.”)

The presentation space position and row/column coordinates are stored in the following two parameters:

Type of Data	DataLength	PSPosition_ReturnCode
Presentation space position	Not applicable (0)	Presentation space position
Row/column coordinates	Row number	Column number

See Also

Connect Presentation Space (1)

Connect Window Services (101)

The Connect Window Services function connects your application and a presentation space so that your application can call the following functions:

- Disconnect Window Services (102)
- Query Window Coordinates (103)
- Window Status (104)

Prerequisites

You must associate a session file name (.ADP) with a short name using either Accessory Manager or the Associate Profile (911) function.

Syntax

```
WinHLLAPI(FunctionNumber,DataString,DataLength,
   PSPosition_ReturnCode)
```

Call Parameters

Parameter	Value
FunctionNumber	CONNECTWINDOWSERVICES (101)
DataString	The short name of the session to connect with.
DataLength	Not applicable; a length of 1 byte is implied.
PSPosition_ReturnCode	Not applicable.

Return Parameters

Parameter	Value	Description
DataString	Not applicable.	Not applicable.
DataLength	Not applicable.	Not applicable.
PSPosition_ReturnCode	WHLLOK (0)	Function succeeded.
	WHLLNOTCONNECTED (1)	An invalid short name was specified.
	WHLLSYSERROR (9)	The function failed due to a system error.
	WHLLUNAVAILABLE (11)	The specified presentation space is already in use.

Comments

This function runs Accessory Manager in the minimized state (if it is not already running), opens the specified session (if it is not already open), and connects to that session.

However, this function cannot be used as a substitute for Connect Presentation Space (1). Functions that require Connect Presentation Space (1) as a prerequisite will return an error if that function is not called. This function exists only to support the Disconnect Window Services (102), Query Window Coordinates (103), and Window Status (104) functions.

See Also

Disconnect Window Services (102), Query Window Coordinates (103), Window Status (104), Associate Profile (911)

Disconnect Window Services (102)

The Disconnect Window Services function disconnects your application from the presentation space that you connected to using the Connect Window Services (101) function.

Prerequisites

Connect Window Services (101)

Syntax

```
WinHLLAPI(FunctionNumber,DataString,DataLength,
   PSPosition_ReturnCode)
```

Call Parameters

Parameter	Value
FunctionNumber	DISCONNECTWINDOWSERVICES (102)
DataString	The short name of the session, or a space or null character for the current session.
DataLength	Not applicable; a length of 1 byte is implied.
PSPosition_ReturnCode	Not applicable.

Return Parameters

Parameter	Value	Description
DataString	Not applicable.	Not applicable.
DataLength	Not applicable.	Not applicable.
PSPosition_ReturnCode	WHLLOK (0)	Function succeeded.
	WHLLNOTCONNECTED (1)	An invalid short name was specified, or the Connect Window Services (101) function was not called prior to calling this function.
	WHLLSYSERROR (9)	The function failed due to a system error.

Comments

Once this function has been called, your application can no longer call the Query Window Coordinates (103) or Window Status (104) functions.

Your application should call this function for each presentation space that has been connected using the Connect Window Services (101) function before closing.

See Also

Connect Window Services (101), Query Window Coordinates (103), Window Status (104)

Query Window Coordinates (103)

The Query Window Coordinates function returns the coordinates of the specified session window or of Accessory Manager's application window.

Prerequisites

Connect Window Services (101)

Syntax

```
WinHLLAPI(FunctionNumber,DataString,DataLength,
   PSPosition_ReturnCode)
```

Call Parameters

Parameter	Value
FunctionNumber	QUERYWINDOWCOORDINATES (103)
DataString	A 17-byte data string. The first byte is the short name of the session, a space or null character for the current session, or an asterisk (*) for the Accessory Manager application window coordinates rather than the session window coordinates. Bytes 2–17 are reserved for returned data.
DataLength	Not applicable; a length of 17 bytes is implied.
PSPosition_ReturnCode	Not applicable.

Return Parameters

Parameter	Value	Description
DataString	A 17-byte data string.	See “Comments.”
DataLength	Not applicable.	Not applicable.
PSPosition_ReturnCode	WHLLOK (0)	Function succeeded.
	WHLLNOTCONNECTED (1)	An invalid short name was specified, or the Connect Window Services (101) function was not called prior to calling this function.
	WHLLPSENDED (12)	The session was closed.

Comments

The returned *DataString* is a 17-byte data string that contains the following information:

Byte	Description
1	The same data sent by the call parameter
2–5	A 32-bit integer (in pixels) of the left X coordinate of the window relative to the desktop
6–9	A 32-bit integer (in pixels) of the bottom Y coordinate of the window relative to the desktop
10–13	A 32-bit integer (in pixels) of the right X coordinate of the window relative to the desktop
14–17	A 32-bit integer (in pixels) of the top Y coordinate of the window relative to the desktop

All X and Y coordinates are given in pixels and are relative to the desktop.

See Also

Connect Window Services (101), Disconnect Window Services (102), Window Status (104)

Window Status (104)

The Window Status function lets your application query or change the size, location, or visible state of the specified session window or of Accessory Manager's application window.

Prerequisites

Connect Window Services (101)

Syntax

```
WinHLLAPI(FunctionNumber,DataString,DataLength,
   PSPosition_ReturnCode)
```

Call Parameters

Parameter	Value
FunctionNumber	WINDOWSTATUS (104)
DataString	A 20-byte string. See "Comments" for details.
DataLength	Not applicable; the default is 16 bytes when the NOHWND104 session parameter is set or 20 bytes when the HWND104 session parameter is set.
PSPosition_ReturnCode	Not applicable.

Return Parameters

Parameter	Value	Description
DataString	A 20-byte string.	See "Comments" for details.
DataLength	Not applicable.	Not applicable.

Parameter	Value	Description
PSPosition_ReturnCode	WHLLOK (0)	Function succeeded.
	WHLLNOTCONNECTED (1)	An invalid short name was specified, or the Connect Window Services (101) function was not called prior to this function.
	WHLLPARAMETERERROR (2)	An invalid parameter was specified.
	WHLLSYSERROR (9)	The function failed due to a system error.
	WHLLPSENDED (12)	The session was closed.

Comments

The *DataString* in the call parameter is a 20-byte string that contains the following information:

Call Parameter DataString

Byte	Description
1	The short name of the session, a space or null character for the current session, or an asterisk (*) for the Accessory Manager application window coordinates rather than the session window coordinates
2	To query for the status, set this to WHLL_WINDOWSTATUS_QUERY. To set the status, set this to WHLL_WINDOWSTATUS_SET.

Call Parameter DataString, continued

Byte	Description
3–4	If byte 2 is query for status, set these bytes to WHLL_WINDOWSTATUS_NULL. If byte 2 is set status, set these bytes to one or more of the following values: • WHLL_WINDOWSTATUS_SIZE—resize the window (invalid with minimize, maximize, restore, or move) • WHLL_WINDOWSTATUS_MOVE—move the window (invalid with minimize, maximize, size, or restore) • WHLL_WINDOWSTATUS_ZORDER—place the window in a specified layer of the display • WHLL_WINDOWSTATUS_SHOW—make the window visible • WHLL_WINDOWSTATUS_HIDE—make the window invisible • WHLL_WINDOWSTATUS_ACTIVATE—activate the window (uses the ZORDER setting or sets focus to the window and places it in the foreground. Used with the session window, it also activates the Accessory Manager application window. • WHLL_WINDOWSTATUS_DEACTIVATE—deactivate the window (uses the ZORDER setting or places it on the bottom) • WHLL_WINDOWSTATUS_MINIMIZE—minimize the window (invalid with maximize, restore, size, or move) • WHLL_WINDOWSTATUS_MAXIMIZE—maximize the window (invalid with minimize, restore, size, or move). This sets focus to the window except when the session window is being maximized and the application window is minimized. • WHLL_WINDOWSTATUS_RESTORE—restore the window (invalid with maximize, minimize, size, or move)
5–6	If byte 2 is query for status or if you are not moving or sizing the window, leave these bytes blank. If byte 2 is set status and you are moving or sizing the window, set these bytes to the X coordinate of the upper left corner of the window.
7–8	If byte 2 is query for status or if you are not moving or sizing the window, leave these bytes blank. If byte 2 is set status and you are moving or sizing the window, set these bytes to the Y coordinate of the upper left corner of the window.

Call Parameter DataString, continued

Byte	Description
9–10	If byte 2 is query for status or if you are not moving or sizing the window, leave these bytes blank. If byte 2 is set status and you are moving or sizing the window, set these bytes to the width of the window.
11–12	If byte 2 is query for status or if you are not moving or sizing the window, leave these bytes blank. If byte 2 is set status and you are moving or sizing the window, set these bytes to the height of the window.
13–16	If byte 2 is query for status or if you are not using ZORDER, leave these bytes blank. If byte 2 is set status and you are using ZORDER, set these bytes to one of these values: • WHLL_WINDOWSTATUS_FRONT-place the window in the foreground • WHLL_WINDOWSTATUS_BACK-place the window in the background
17–20	Reserved for returned data

The returned *DataString* is a 20-byte string that contains the following information:

Return Parameter DataString

Byte	Description
1–2	The same data sent by the call parameter
3–4	If byte 2 is set status, these bytes are the same as the call parameter. If byte 2 is a query for status, the following are the possible return values (you can combine more than one status using a binary OR operation): • WHLL_WINDOWSTATUS_SHOW—the window is visible. • WHLL_WINDOWSTATUS_HIDE—the window is invisible. • WHLL_WINDOWSTATUS_ACTIVATE—the window is activated. • WHLL_WINDOWSTATUS_DEACTIVATE—the window is deactivated. • WHLL_WINDOWSTATUS_MINIMIZE—the window is minimized. • WHLL_WINDOWSTATUS_MAXIMIZE—the window is maximized.
5–6	If byte 2 is query for status, these bytes are the X coordinate of the upper left corner of the window. If byte 2 is set status, these bytes are the same as the call parameter.
7–8	If byte 2 is query for status, these bytes are the Y coordinate of the upper left corner of the window. If byte 2 is set status, these bytes are the same as the call parameter.
9–10	If byte 2 is query for status, these bytes are the width of the window. If byte 2 is set status, these bytes are the same as the call parameter.
11–12	If byte 2 is query for status, these bytes are the height of the window. If byte 2 is set status, these bytes are the same as the call parameter.
13–16	The same data sent by the call parameter
17–20	If byte 2 is set status, or if byte 2 is query for status and the NOHWND104 session parameter is set, these bytes are blank. If byte 2 is query for status and the HWND104 session parameter is set, the application returns a 32-bit string containing the window handle.

If you set the size and position of a session window and Autosize Window is selected in the session configuration, the resulting size and position might be slightly different from what you specified. When Autosize Window is selected, the session window automatically resizes based on the size of the font. To determine the exact window size and position, call the Window Status (104) function again, this time querying for the window status rather than setting it.

For Accessory Manager's application window, the X and Y coordinates are relative to the whole desktop.

For session windows, the X and Y coordinates are relative to Accessory Manager's application window and include the title bar, border, and scroll bars, if present. To get coordinates relative to the whole desktop, use the Query Window Coordinates (103) function.

See Also

Connect Window Services (101), Disconnect Window Services (102), Query Window Coordinates (103)

Associate Profile (911)

The Associate Profile function associates the specified short name with the specified session file name.

Prerequisites

The specified session must already exist.

Syntax

```
WinHLLAPI(FunctionNumber,DataString,DataLength,
   PSPosition_ReturnCode)
```

Call Parameters

Parameter	Value
FunctionNumber	PROFILEASSOCIATE (911)
DataString	A string up to 255 bytes. The first bye is the short name to associate with the session file name. The remaining bytes are the file name of the session to associate with the specified short name.
DataLength	Not applicable.
PSPosition_ReturnCode	Not applicable.

Return Parameters

Parameter	Value	Description
DataString	Not applicable.	Not applicable.
DataLength	Not applicable.	Not applicable.
PSPosition_ReturnCode	WHLLOK (0)	Function succeeded.
	WHLLPARAMETERERROR (2)	An invalid parameter was passed.
	WHLLSYSERROR (9)	The function failed due to a system error.

Comments

If the short name is already associated with a session file name, this function changes the association to the specified file name.

See Also

Remove Profile (912), Get Associations (913), Find File Name (914), Find Short Name (915)

Remove Profile (912)

The Remove Profile function removes any session file name from the specified short name.

Prerequisites

None.

Syntax

```
WinHLLAPI(FunctionNumber,DataString,DataLength,
   PSPosition_ReturnCode)
```

Call Parameters

Parameter	Value
FunctionNumber	PROFILEREMOVE (912)
DataString	The short name that you want to remove from any session file name.
DataLength	Not applicable.
PSPosition_ReturnCode	Not applicable.

Return Parameters

Parameter	Value	Description
DataString	Not applicable.	Not applicable.
DataLength	Not applicable.	Not applicable.
PSPosition_ReturnCode	WHLLOK (0)	Function succeeded.
	WHLLPARAMETERERROR (2)	An invalid short name was specified.
	WHLLSYSERROR (9)	The function failed due to a system error.
	WHLLNOMATCH (42)	No session associated with the short name.

Comments

If you call this function and then call another function using the removed short name, that function will return an error.

See Also

Association Profile (911), Get Associations (913), Find File Name (914), Find Short Name (915)

Get Associations (913)

The Get Associations function retrieves a list of all the short names that have been associated with session file names.

Prerequisites

None.

Syntax

```
WinHLLAPI(FunctionNumber,DataString,DataLength,
   PSPosition_ReturnCode)
```

Call Parameters

Parameter	Value
FunctionNumber	PROFILEGETASSOCIATIONS (913)
DataString	A data string for the returned value (at least 27 bytes long).
DataLength	Not applicable.
PSPosition_ReturnCode	Not applicable.

Return Parameters

Parameter	Value	Description
DataString	A null-terminated string of short names that are associated with session file names.	If no session file names are associated with short names, this parameter contains a null character.
DataLength	Not applicable.	Not applicable.
PSPosition_ReturnCode	WHLLOK (0)	Function succeeded.
	WHLLSYSERROR (9)	The function failed due to a system error.

Comments

None.

See Also

Association Profile (911), Remove Profile (912), Find File Name (914), Find Short Name (915)

Find File Name (914)

The Find File Name function retrieves the session file name associated with a specified short name.

Prerequisites

None.

Syntax

```
WinHLLAPI(FunctionNumber,DataString,DataLength,
   PSPosition_ReturnCode)
```

Call Parameters

Parameter	Value
FunctionNumber	PROFILEFINDFILENAME (914)
DataString	The short name for which you want to retrieve the session file name.
DataLength	The length of the *DataString* that will be returned (256 bytes recommended).
PSPosition_ReturnCode	Not applicable.

Return Parameters

Parameter	Value	Description
DataString	The session file name associated with the short name specified in the call parameter.	The returned *DataString* includes the full DOS path to the session file.
DataLength	Not applicable.	Not applicable.
PSPosition_ReturnCode	WHLLOK (0)	Function succeeded.
	WHLLPARAMETERERROR (2)	An invalid short name was specified.
	WHLLSYSERROR (9)	The function failed due to a system error.
	WHLLNOMATCH (42)	The specified short name is not associated with a session file name.

Comments

None.

See Also

Association Profile (911), Remove Profile (912), Get Associations (913), Find Short Name (915)

Find Short Name (915)

The Find Short Name function retrieves the short name associated with a specified session file name.

Prerequisites

None.

Syntax

```
WinHLLAPI(FunctionNumber,DataString,DataLength,
   PSPosition_ReturnCode)
```

Call Parameters

Parameter	Value
FunctionNumber	PROFILEFINDSHORTNAME (915)
DataString	The session file name that you want to retrieve the short name of. Be sure to include the drive and folder where the session file is located, as well as the file extension (for example, C:\INFOCN32\ACCMGR32\SESSION1.ADP).
DataLength	Not applicable.
PSPosition_ReturnCode	Not applicable.

Return Parameters

Parameter	Value	Description
DataString	The short name associated with the session file name specified in the call parameter.	Not applicable.
DataLength	Not applicable.	Not applicable.

Parameter	Value	Description
PSPosition_ReturnCode	WHLLOK (0)	Function succeeded.
	WHLLPARAMETERERROR (2)	An invalid session file name was specified.
	WHLLSYSERROR (9)	The function failed due to a system error.
	WHLLNOMATCH (42)	The specified session file name is not associated with a short name.

Comments

None.

See Also

Association Profile (911), Remove Profile (912), Get Associations (913), Find File Name (914)

Return Codes A

In This Appendix This appendix includes the following headings:

IHLLAPI Return Codes

The following table lists the standard return codes, their causes, and suggested solutions:

Return Code	**Explanation**
WHLLOK (0)	The function completed successfully.
WHLLNOTCONNECTED (1)	The application is not connected to a session, but tried to execute a function that requires a connection. Call the Connect Presentation Space (1) function before calling a function that requires a connection.
WHLLPARAMETERERROR (2)	An invalid parameter was passed, or the *DataLength* was too short. Double-check the parameters for the function. If the parameter is valid and is being passed as a variable, make sure that the variable is being changed before it is passed to the function. If the EAB session parameter is set, you must provide a *DataString* at least twice as long as the size of the visible data.
WHLLFTXCOMPLETE (3)	The function succeeded.
WHLLPSBUSY (4)	The presentation space is busy or input is inhibited. Your application should wait until the session is ready to accept data. If the function did not try to send data to the presentation space (for example, if the function was Connect Presentation Space), the function otherwise completed successfully, and you can ignore this return code.
WHLLINHIBITED (5)	Your application tried to send a keystroke before the session was ready for input. To prevent this error, your application must wait for the session to become ready for input. Once the session is ready, call the function again. For information on waiting for host responses, refer to “Waiting for Host Responses” on page 27.

Return Code	Explanation
WHLLTRUNCATED (6)	The application specified a DataLength shorter than the presentation space. Check the DataLength. If the length is being passed as a variable, make sure that the variable contains the correct data when it is passed to the function.
WHLLPOSITIONERROR (7)	The application specified a presentation space position of zero or a number larger than the number of positions in the presentation space. Specify a value greater than zero, or use the Query Sessions (10) function to determine the size of the presentation space.
WHLLNOTAVAILABLE (8)	The application called a function that is not supported in this version of IHLLAPI, or the application did not call the Start Host Notification (23) function prior to calling Query Host Update (24) or Stop Host Notification (25). Verify that the function is supported, and be sure to call any prerequisite functions.
WHLLSYSERROR (9)	A system error occurred. Your application should disconnect from any connected sessions, call Reset System (21) to restore the default session parameters, then close.
WHLLUNAVAILABLE (11)	The presentation space is unavailable or already in use, usually because another IHLLAPI application has connected to the specified session. Use Query Sessions (10) to find valid sessions and connect to the next available session, or close with an error message if no more sessions are available.
WHLLPSENDED (12)	The session was closed. Close your application or prompt the user to reopen the session and try again.
WHLLNOFIELD (24)	The presentation space is unformatted, the field was not found, or the host returned data that the application did not expect. Verify that the presentation space is formatted, or substitute a function that interacts with the presentation space rather than a particular field.

Return Code	Explanation
WHLLFTXABORTED (27)	The file transfer timed out due to an error or was manually cancelled by the user.

Functions That Return Standard Return Codes

The following functions return only the standard error codes and can use a standard error handler:

Connect Presentation Space (1)	Query System (20)
Disconnect Presentation Space (2)	Reset System (21)
Send Key (3)	Query Session Status (22)
Wait (4)	Start Host Notification (23)
Copy Presentation Space (5)	Stop Host Notification (25)
Query Cursor Location (7)	Search Field (30)
Copy Presentation Space to String (8)	Find Field Position (31)
Set Session Parameters (9)	Find Field Length (32)
Query Sessions (10)	Copy String to Field (33)
Copy OIA (13)	Copy Field to String (34)
Query Field Attribute (14)	Set Cursor (40)
Copy String to Presentation Space (15)	

The other functions either use the standard return codes differently, or use other return codes in addition to the standard codes. You can handle these functions by checking for special cases first and then calling your standard error handler if the return code is standard.

Note: The Convert Position or RowCol (99) function has unique return codes; you must respond to them outside the standard error handler. Refer to Chapter 3, "IHLLAPI Functions," for more information on this function.

Sample Return Code Usage

The following pseudo-code example shows how to manage the possible return codes for the Send Key (3) function.

This type of return code management can be applied to all IHLLAPI functions. However, be sure to refer to Chapter 3, “IHLLAPI Functions,” for exact return codes and their explanation as they pertain to each particular function.

```
Start SendKey Function
  Function Number = SENDKEY
  Data String = "MyName@E"
  String Length = equals the length of the
    DataString
  PS Position = 0 (Zero)

  HLLAPI CALL(FunctionNumber, DataString,
    DataLength, PSPosition_Return Code)

  Start Return Code Check
    Return Code = WHLLOK: Function was successful.
      Continue with the next function in the
      application.

    Return Code = WHLLNOTCONNECTED: Application
      not connected to a session. Call Query
      Sessions (10) to find a valid short name
      and then call Connect Presentation Space (1)
      using the short name returned from the Query
      Sessions function.
      Once these functions have successfully been
      completed, retry the original SendKey
      function.

    Return Code = WHLLPARAMETERERROR:  Incorrect
      parameter was passed to the function. Close
      the application with an error message
      indicating the nature of the failure.
```

```
          Return Code = WHLLPSBUSY:  Presentation space
            is busy. Do not try to send data to the
            host. Wait until the session is no longer
            busy before executing the next function
            or calling the original function again.

          Return Code = WHLLINHIBITED:  Input to the
            presentation space is inhibited. This
            normally indicates that the keyboard is
            locked and that the session cannot accept
            further input. Wait until this condition
            is cleared, then retry the original
            function.

          Return Code = WHLLSYSERROR:  A system error
            occurred. Call Reset System (21) and close
            the application with an error message
            indicating that a system error occurred.

        End Return Code Check

      End SendKey Function
```

Attribute Values B

In This Appendix

This appendix includes the following headings:

T 27 Attributes

The following table lists the character attribute values that are returned by an T 27 session when the EAB session parameter is set, as well as what each value represents:

Bit Position	Description	Value
7	Reverse video	0 = Normal 1 = Reverse
6	Underscore	0 = None 1 = Underscore
5	Blink	0 = None 1 = Blink
4	Bright	0 = None 1 = Bright
3	Secure	0 = None 1 = Secure
2–0	Reserved	Spaces

UTS Attributes

The following table lists the character attribute values that are returned by a UTS session when the EAB session parameter is set, as well as what each value represents:

Bit Position	Description	Value
7	Reverse video	0 = Normal 1 = Reverse
6	Underscore	0 = None 1 = Underscore
5	Blink	0 = None 1 = Blink
4	Left column separator	0 = None 1 = Column separator
3	Secure	0 = None 1 = Secure
2	Right column separator	0 = None 1 = Column separator
1	Strikethrough	0 = None 1 = Strikethrough
0	Upperscore	0 = None 1 = Upperscore

The following table lists the color attribute values that are returned by a UTS session when the EAB session parameter is set, as well as what each value represents:

Bit Position	Description	Value	
7–4	Background character colors	0000 = Black 0001 = Blue 0010 = Green 0100 = Red 0011 = Cyan	0101 = Magenta 0110 = Yellow 0111 = White
3–0	Foreground character colors	0000 = Black 0001 = Blue 0010 = Green 0011 = Cyan 0100 = Red 0101 = Magenta 0110 = Yellow 0111 = White 1000 = Dark grey	1001 = Light blue 1010 = Light green 1011 = Light cyan 1100 = Light red 1101 = Light magenta 1110 = Brown 1111 = Light grey

Note: These are the colors specified by the host rather than any colors specified by the emulator.

IHLLAPI Header Files

C

In This Appendix

This appendix provides a printed listing of three header files (WHLLAPI.H, HLLAPI.H, and IHLAPI32.H) that are included with the Automation Development Kit. These files must be included with applications written in C, C++, or any other language that can include a header file.

WHLLAPI.H Header File

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

#include "windows.h"

#ifndef WHLLAPIINC
#define WHLLAPIINC

#pragma pack(1) // Pack HLLAPI structures on 1-byte boundary
#pragma message("Information: HLLAPI structures are packed on 1-byte
boundaries.")

/****** 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  /* Msg 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 reqs 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 /* 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

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

#ifdef __cplusplus
   extern "C"
{
#endif

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

extern void WINAPI WinHLLAPI(LPWORD, LPSTR, LPWORD, LPWORD);
extern HANDLE WINAPI WinHLLAPIAsync(HWND, LPWORD, LPSTR, LPWORD, LPWORD);
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(FARPROC);
extern BOOL WINAPI WinHLLAPIUnhookBlockingHook(void);

#ifdef __cplusplus
}
#endif

#pragma pack()      // Revert to previous packing

#endif
```

HLLAPI.H Header File

```
/***************************************************************************
                Module Header
****************************************************************************

File:       hllapi.h
Purpose:    data structures, definitions & prototypes comprising HLLAPI

Notes:
            13-Jun-1991:
            Added new hllapi functions HLL_CopyOIA5 and HLL_CopyOIA9.

            01-Apr-1991:
            HLL_WSCTRLSTARTEMULATOR's EMULATORCONTROL structure byPower field
            Has been renamed byReserved, reflecting the fact that the
            subfunction no longer affects the POWER condition of a new
            terminal emulator.

            30-Oct-1990:
            Added HLL_PROTECTED as synonym for return code 5.

            18-Oct-1990:
            Added WSCTRL subfunctions: -BLOCKEMULATORUPDATES &
             -ENABLEEMULATORUPDATES.

            05-Feb-1990:
            HLL_ReceiveFile and HLL_SendFile: additional parameter.

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


/************************************************************
HLLAPI function selectors ~~ HLLAPI subfunction selectors
************************************************************/
#define HLL_ATMQUERYSYSTEM              0
#define HLL_CONNECTPS                   1
#define HLL_DISCONNECTPS                2
#define HLL_SENDKEY                     3
#define HLL_WAIT                        4

#define HLL_COPYPS                      5
#define HLL_SEARCHPS                    6
#define HLL_QUERYCURSOR                 7
#define HLL_COPYPSTOSTRING              8
#define HLL_SETHLLWINPARAMETERS         9
```

```
#define HLL_QUERYSESSIONS              10
#define HLL_RESERVE                    11
#define HLL_RELEASE                    12
#define HLL_COPYOIA                    13
#define HLL_QUERYFIELDATTRIBUTE        14

#define HLL_COPYSTRINGTOPS             15
#define HLL_WSCTRL                     16
#define HLL_PAUSE                      18
#define HLL_QUERYSYSTEM                20
#define HLL_RESETHLLWIN                21

#define HLL_QUERYSESSIONSTATUS         22
#define HLL_STARTHOSTNOTIFICATION      23
#define HLL_QUERYHOSTUPDATE            24
#define HLL_STOPHOSTNOTIFICATION       25
#define HLL_SEARCHFIELD                30

#define HLL_FINDFIELDPOSITION          31
#define HLL_FINDFIELDLENGTH            32
#define HLL_COPYSTRINGTOFIELD          33
#define HLL_COPYFIELDTOSTRING          34
#define HLL_SETCURSOR                  40

#define HLL_STARTKEYSTROKEINTERCEPT    50
#define HLL_GETKEY                     51
#define HLL_POSTINTERCEPTSTATUS        52
#define HLL_STOPKEYSTROKEINTERCEPT     53
#define HLL_SENDFILE                   90

#define HLL_RECEIVEFILE                91
#define HLL_CONVERT                    99

/***********
sui generis
************/
#define HLL_HLLAPI                     500
#define HLL_ENUMHLLWINS                501
#define HLL_QUERYHLLWINPARAMETERS      502
#define HLL_SETMESSAGELOOPCALLBACK     503
```

```
#ifndef RC_INVOKED
/*************************
system manifest constants
*************************/
#define MAXSENDKEYLENGTH                255
#define MAXPAUSEDURATION                0xFFFF
#define MINKEYSTROKEBUFFERLENGTH        6
#define MAXFILETRANSFERSTRINGLENGTH     512     /*  07-Jun-1990 */

/*********************************
informal list of HLL return codes
*********************************/
#define HLL_SUCCESS                    0
#define HLL_INVALIDPSID                1
#define HLL_INVALIDPARAMETER           2
#define HLL_SESSIONOCCUPIED            4
#define HLL_TIMEOUT                    4 /*  e.g. Wait (XCLOCK/XSYSTEM)*/
#define HLL_PSLOCKED                   5 /*  e.g. ConnectPS  */
#define HLL_PROTECTED                  5
#define HLL_FIELDSIZEMISMATCH          6 /*  e.g. CopyFieldToString  */
#define HLL_DATATRUNCATED              6 /*  e.g. CopyStringToField  */
#define HLL_INVALIDPSPOSITION          7
#define HLL_NOPRIORSTARTKEYSTROKE      8
#define HLL_NOPRIORSTARTHOSTNOTIFY     8
#define HLL_SYSTEMERROR                9
#define HLL_RESOURCEUNAVAILABLE        11
#define HLL_SEARCHSTRINGNOTFOUND       24 /*  e.g. SearchField    */
#define HLL_UNFORMATTEDHOSTPS          24 /*  e.g. CopyFieldToString  */
#define HLL_NOSUCHFIELD                24 /*  e.g. FindFieldPosition  */
#define HLL_NOHOSTSESSIONUPDATE        24 /*  new code for QueryHostUpdate */
#define HLL_KEYSTROKESNOTAVAILABLE     25
#define HLL_HOSTSESSIONUPDATE          26 /*  end of Pause    */
#define HLL_KEYSTROKEQUEUEOVERFLOW     31
#define HLL_MEMORYUNAVAILABLE          101 /*  cf StartKSIntercept   */
#define HLL_DELAYENDEDBYCLIENT         102 /*  Wait, Pause, GetKey */
#define HLL_UNCONFIGUREDPSID           103 /* Connect, Intercept    */
#define HLL_NOEMULATORATTACHED         104 /*  keystroke intercept, etc */
#define HLL_WSCTRLFAILURE              105

/********************************
values for Post Intercept Status
********************************/
#define HLL_INTERCEPTACCEPTED           0
#define HLL_INTERCEPTREJECTED           1
```

```
/*********************************
wOptionCode values for HLL_GetKey
*********************************/
#define HLL_GETKEYASCII                 0
#define HLL_GETKEYMNEMONIC              1
#define HLL_GETKEYSHIFTED               2

/*************************************
options for HLL_StartHostNotification
*************************************/
#define HLL_NOTIFYALLUPDATES            127
#define HLL_NOTIFYPSUPDATE              1
#define HLL_NOTIFYOIAUPDATE             2
#define HLL_NOTIFYCURSORUPDATE          4
#define HLL_NOTIFYBEEP                  8
#define HLL_NOTIFYBASECOLORCHANGE       16
#define HLL_NOTIFYMODELCHANGE           32
#define HLL_NOTIFYPOWERCHANGE           64

/*************************************************
field specifiers for HLL_FindFieldLength/Position
*************************************************/
#define HLL_THISFIELD                   0
#define HLL_NEXTFIELD                   1
#define HLL_PREVIOUSFIELD               2
#define HLL_NEXTPROTECTEDFIELD          3
#define HLL_NEXTUNPROTECTEDFIELD        4
#define HLL_PREVIOUSPROTECTEDFIELD      5
#define HLL_PREVIOUSUNPROTECTEDFIELD    6

/****************************************
session specifiers for HLL_QuerySessions
****************************************/
#define HLL_QUERYSESSIONSCONFIGURED    1   /*  in config file  */
#define HLL_QUERYSESSIONSOPENED        2   /*  opened, i.e. OpenSession () */
#define HLL_QUERYSESSIONSPOWERED       3   /*  terminal on */  /*  default */
```

```
/********************************
option specifiers for HLL_WSCtrl
********************************/
#define HLL_WSCTRLOPENCONFIGURATION     1
#define HLL_WSCTRLCLOSECONFIGURATION    2
#define HLL_WSCTRLQUERYCONFIGURATION    3
#define HLL_WSCTRLOPENLAYOUT            4
#define HLL_WSCTRLQUERYLAYOUT           5
#define HLL_WSCTRLEMULATORHANDLE        6
#define HLL_WSCTRLSTARTEMULATOR         7
#define HLL_WSCTRLSTOPEMULATOR          8
#define HLL_WSCTRLTERMINALON            9
#define HLL_WSCTRLTERMINALOFF           10
#define HLL_WSCTRLSETEXECUTEPATH        11
#define HLL_WSCTRLQUERYEXTRADIRECTORY   12
#define HLL_WSCTRLALLOWEMULATORUPDATES  13
#define HLL_WSCTRLBLOCKEMULATORUPDATES  14

typedef struct tagEmulatorControl {
   char    cPSID;          /*  minimum requirement */
   BYTE    byVisibility;   /*  N(ormal) I(conic) M(aximized) H(idden)  */
   BYTE    byReserved;     /*  until 01-Apr-1991, byPower  */
   BYTE    byCase;         /*  U(pper) M(ixed) */
   WORD    wLeft;          /*  absolute position, in screen coordinates    */
   WORD    wBottom;
   WORD    wRight;
   WORD    wTop;
   }       EMULATORCONTROL, * LPEMULATORCONTROL, * NPEMULATORCONTROL;
    /******************************************************************
    19-Jul-1989 this structure is subject to expansion: e.g. fonts, ...
    ******************************************************************/
```

```
/*********************************
option specifiers for HLL_Convert
*********************************/
#define HLL_CONVERTPOSITION             1
#define HLL_CONVERTROWCOLUMN            2

typedef struct tagHLLParams {
    BYTE    byAttribute;        /*  ATTRB or NOATTRB    */
    BYTE    byAutoReset;        /*  AUTORESET or NOAUTORESET    */
    BYTE    byConnectType;      /*  CONLOG or CONPHYS   */
    BYTE    byEAB;              /*  EAB or NOEAB    */
    char    cEscape;            /*  default '@' */
    BYTE    byPause;            /*  IPAUSE or FPAUSE    */
    BYTE    bySearchOrigin;     /*  SRCHALL or SRCHFROM */
    BYTE    bySearchDirection;  /*  SRCHFORWARD or SRCHBKWD */
    WORD    wTimeOut;           /*  0..64k          */
    BYTE    byTrace;            /*  TRON or TROFF   */
    BYTE    byWait;             /*  TWAIT, LWAIT, NWAIT */
    BYTE    byXlate;            /*  ASCII<->3270DC translation or not   */
    }       HLLPARAMS, * NPHLLPARAMS, * LPHLLPARAMS;

/******************************
values for fields of HLLPARAMS
******************************/

/******************
byAttribute values
******************/
#define HLLWIN_ATTRB            1
#define HLLWIN_NOATTRB          2

/******************
byAutoReset values
******************/
#define HLLWIN_AUTORESET        1
#define HLLWIN_NOAUTORESET      2

/***********************
byConnectionType values
***********************/
#define HLLWIN_CONLOG           1
#define HLLWIN_CONPHYS          2

/************
byEAB values
************/
#define HLLWIN_EAB              1
#define HLLWIN_NOEAB            2
/*************
byPause values
*************/
```

```
#define HLLWIN_FPAUSE           1
#define HLLWIN_IPAUSE           2

/*********************
bySearchOrigin values
*********************/
#define HLLWIN_SRCHALL          1
#define HLLWIN_SRCHFROM         2

/************************
bySearchDirection values
************************/
#define HLLWIN_SRCHFRWD         1
#define HLLWIN_SRCHBKWD         2

/**************
byTrace values
**************/
#define HLLWIN_TRON             1
#define HLLWIN_TROFF            2

/************
byWait values
************/
#define HLLWIN_TWAIT            1
#define HLLWIN_LWAIT            2
#define HLLWIN_NWAIT            3

/**************
byXlate values
**************/
#define HLLWIN_XLATE            1
#define HLLWIN_NOXLATE          2

typedef struct tagATMSystem {
    WORD    wHLLAPIVersionNumber;
    WORD    wHLLAPILevelNumber;
    }       ATMSYSTEM, * NPATMSYSTEM, * LPATMSYSTEM;

typedef struct tagStartIntercept {
    char    cPSID;
    WORD    wKeyFilter;
    WORD    wQueueLength;
    BOOL    bWindowsMessage;
    }       STARTINTERCEPT, * NPSTARTINTERCEPT, * LPSTARTINTERCEPT;
/**********************************
values for wKeyFilter interception
**********************************/
#define HLL_INTERCEPTAIDKEYS    1
#define HLL_INTERCEPTALLKEYS    2
```

```
typedef struct tagKeystroke {
    char    cPSID;
    char    szKeystroke [MINKEYSTROKEBUFFERLENGTH];
    }       KEYSTROKE, * NPKEYSTROKE, * LPKEYSTROKE;

typedef struct tagOIAGroup {
    char    szOIAGroup [22];
    }       OIAGROUP, * NPOIAGROUP, * LPOIAGROUP;

typedef struct tagSessions {
    char    cPSID;
    char    szLongName [8];
    BYTE    byState;
            /*******************************************
            bit 0:  configured?         bit 1:  opened?
            bit 2:  powered?
            *******************************************/
    WORD    wPSSize;
    }       SESSIONS, * NPSESSIONS, * LPSESSIONS;

typedef struct tagSessionStatus {
    char    cPSID;  /*  session short name  */
    char    szLongName [8];
    BOOL    bType;   /*  FALSE = CUT;    TRUE = DFT   */
    BYTE    byCharacteristics;
            /****************************************************
            bit 0: EABs? Y/N        bit 1: Programmed Symbols Y/N
            ****************************************************/
    BYTE    byUsage;
            /***************************************************
            bit 0: Configured? Y/N  bit 1: Opened? Y/N
            bit 2: Powered? Y/N     bit 3: AutoPowered? Y/N
            bit 4: Emulated? Y/N    bit 5: HLLAPI-Connected? Y/N
            bit 6: FileTrans? Y/N
            ***************************************************/
            /******************************************************************
              note that bit 5, HLLAPI-Connected, may be either HLLWin connection,
              HLLWin keystroke interception, or HLLWin-initiated file transfer
            ******************************************************************/

    WORD    wRows;
    WORD    wColumns;
    }       SESSIONSTATUS, * NPSESSIONSTATUS, * LPSESSIONSTATUS;

/*****************************************************************
definitions for use in probing the bySessionCharacteristics byte
*****************************************************************/
#define HLL_SESSIONEABS                 0x01
#define HLL_SESSIONPROGRAMMEDSYMBOLS    0x02

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

```
definitions for use in probing the bySessionUsage byte in SessionStatus
**********************************************************************/
#define HLL_SESSIONCONFIGURED           0x01
#define HLL_SESSIONOPENED               0x02
#define HLL_SESSIONPOWERED              0x04
#define HLL_SESSIONAUTOPOWERED          0x08
#define HLL_SESSIONEMULATED             0x10
#define HLL_SESSIONCONNECTED            0x20    /*****************************
                                                HLLWin connections, keystroke
                                                interception, or monitors
                                                *****************************/
#define HLL_SESSIONFILETRANSFER         0x40

typedef struct tagSystem {
    BYTE    byHLLAPIMonth;
    BYTE    byHLLAPIDay;
    WORD    wHLLAPIYear;
    char    cPSID;
    DWORD   dwSystemError;
    }       SYSTEM, * NPSYSTEM, * LPSYSTEM;

/***************************************
exported routines, in alphabetic order
***************************************/

#ifdef __cplusplus
   extern "C"
   {
#endif

WORD WINAPI
HLL_AttachmateQuerySystem (
    HWND        hWnd,
    LPATMSYSTEM lpATMSystem);
```

```
WORD WINAPI
HLL_ConnectPS (
    HWND        hWnd,
    char        cPSID);

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

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

WORD WINAPI
HLL_CopyOIA (                         // Xclock info switchable between 5/9
    HWND        hWnd,
    LPSTR       lpOIA);               /*  103 chars of binary data    */

WORD WINAPI
HLL_CopyOIA9 (                        // Xclock info in position 9
    HWND        hWnd,
    LPSTR       lpOIA);               /*  103 chars of binary data    */

WORD WINAPI
HLL_CopyOIA5 (                        // Xclock info in position 5
    HWND        hWnd,
    LPSTR       lpOIA);               /*  103 chars of binary data    */

WORD WINAPI
HLL_CopyPS (
    HWND        hWnd,
    LPSTR       lpBuffer);

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

WORD WINAPI
HLL_CopyStringToPS (
    HWND        hWnd,
    LPSTR       lpBuffer,
    WORD        wBufferLength,
    WORD        wPSP);

WORD WINAPI
HLL_DisconnectPS (
    HWND        hWnd);

DWORD WINAPI
HLL_EnumHLLWins (
    HWND        hWnd,
    HWND       *nWnd);

DWORD WINAPI
HLL_FindFieldLength (
    HWND        hWnd,
    WORD        wFieldSpecifier,
    WORD        wTargetFieldPSP);

DWORD WINAPI
HLL_FindFieldPosition (
    HWND        hWnd,
    WORD        wFieldSpecifier,
    WORD        wTargetFieldPSP);

WORD WINAPI
HLL_GetKey (
    HWND        hWnd,
    LPKEYSTROKE lpKeystroke);

WORD WINAPI
HLL_Pause (
    HWND        hWnd,
    WORD        wDuration);     /*  500-millisecond units   */
WORD WINAPI
HLL_PostInterceptStatus (
    HWND        hWnd,
    char        cPSID,
    WORD        wStatus);

WORD WINAPI
HLL_QueryHLLWinParameters (
    HWND        hWnd,
    LPHLLPARAMS lpHLLParams);

DWORD WINAPI
HLL_QueryCursor (
    HWND        hWnd);

DWORD WINAPI
HLL_QueryFieldAttribute (
    HWND        hWnd,
    WORD        wPSP);

DWORD WINAPI
HLL_QueryHostUpdate (
    HWND        hWnd,
    char        cPSID);

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

WORD WINAPI
HLL_QuerySessionStatus (
    HWND            hWnd,
    LPSESSIONSTATUS lpSessionStatus);

WORD WINAPI
HLL_QuerySystem (
    HWND        hWnd,
    LPSYSTEM    lpSystem);

DWORD WINAPI
HLL_ReceiveFile (
    HWND        hWnd,
    LPSTR       lpszReceiveCommand,
    LPWORD      lpwSequenceID);

WORD WINAPI
HLL_Release (
    HWND        hWnd);
```

```
WORD WINAPI
HLL_Reserve (
    HWND        hWnd);

WORD WINAPI
HLL_ResetHLLWin (
    HWND        hWnd);

DWORD WINAPI
HLL_SearchField (
    HWND        hWnd,
    LPSTR       lpsSearchString,
    WORD        wStringLength,      /*  if HLLWIN_NOXLATE   */
    WORD        wPSP);              /*  if HLLWIN_SRCHFROM  */

DWORD WINAPI
HLL_SearchPS (
    HWND        hWnd,
    LPSTR       lpsSearchString,
    WORD        wStringLength,      /*  if HLLWIN_NOXLATE   */
    WORD        wPSP);              /*  either HLLWIN_SRCHALL or SRCHFROM   */

DWORD WINAPI
HLL_SendFile (
    HWND        hWnd,
    LPSTR       lpszReceiveCommand,
    LPWORD      lpwSequenceID);

WORD WINAPI
HLL_SendKey (
    HWND        hWnd,
    LPSTR       lpszKeys);

WORD WINAPI
HLL_SetCursor (
    HWND        hWnd,
    WORD        wCursorLocation);

WORD WINAPI
HLL_SetMessageLoopCallback (
    HWND        hWnd,
    FARPROC     lpfnCallback);
```

```
WORD WINAPI
HLL_SetHLLWinParameters (
    HWND        hWnd,
    LPHLLPARAMS lpHLLParams);

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

WORD WINAPI
HLL_StartKeystrokeIntercept (
    HWND                hWnd,
    LPSTARTINTERCEPT    lpIntercept);

WORD WINAPI
HLL_StopHostNotification (
    HWND        hWnd,
    char        cPSID);

WORD WINAPI
HLL_StopKeystrokeIntercept (
    HWND        hWnd,
    char        cPSID);

WORD WINAPI
HLL_Wait (
    HWND        hWnd);

WORD WINAPI
HLL_WSCtrl (
    HWNDhWnd,
    WORD        wOption,
    LPVOID      lpvState,
    WORD        wStateLength);

#ifdef __cplusplus
   }
#endif

#endif  /*  RC_INVOKED  */
```

IHLAPI32.H Header File

```
//INCLUDE FILES

#ifdef __cplusplus
   #include "afxwin.h"
#endif

// to allow exporting the DLL functions

#ifdef VOY32DLL
   #define VOYAPI __declspec(dllexport)
#else
   #define VOYAPI __declspec(dllimport)
#endif

//constants

//Global variables

// Function prototypes

#ifndef IHLLAPIINC
#define IHLLAPIINC

#pragma pack(1)       // Pack HLLAPI structures on 1-byte boundary
#pragma message("Information: HLLAPI structures are packed on 1-byte
boundaries.")

/****** Function numbers ***********************************************/
#define    PROFILEASSOCIATE            911
#define    PROFILEREMOVE               912
#define    PROFILEGETASSOCIATIONS      913
#define    PROFILEFINDFILENAME         914
#define    PROFILEFINDSHORTNAME        915
/****** Windows HLLAPI structure ***************************************/

#define WHLLDESCRIPTION_LEN 127

typedef  struct   tagIHLLAPIDATA {
         WORD     wVersion;
         char     szDescription[WHLLDESCRIPTION_LEN+1];
} IHLLAPIDATA, * PIHLLAPIDATA, FAR * LPIHLLAPIDATA;

#pragma pack()      // Revert to previous packing

#endif
```

Troubleshooting D

In This Appendix This appendix contains the following sections:

General Troubleshooting Procedures

If you have problems running your IHLLAPI applications, follow these steps:

1. Make sure that IHLAPI32.DLL is accessible. IHLAPI32.DLL must be in the Windows PATH.

2. Make sure that Accessory Manager is accessible. If you use the Connect Presentation Space (1) function to run Accessory Manager and open a session, Accessory Manager must be in the Windows PATH.

3. Make sure that the appropriate session file name is associated with the appropriate short name. Refer to "Using Short Names" on page 20 for more information.

4. Check the prerequisites for each function. Some functions have prerequisite functions that must be called prior to calling that function.

5. Check the function using the test application included with the ADK. (For more information, refer to "Trying Out IHLLAPI Functions" on page 15.) If the function works with the test application but not with your IHLLAPI application, the problem might be in your IHLLAPI application.

6. Review the return code generated by the function. Return codes are provided for each function in the PSPosition_ReturnCode portion of the "Return Parameters" section. In addition, refer to Appendix A, "Return Codes," for information about standard return codes and return code handling.

7. Consult your distributor. If you cannot identify and solve the problem without assistance, contact your product distributor. Call from a location where you have access to the problem PC.

RAWMODE E

In This Appendix

This appendix explains the use of raw mode when sending host data. The following sections are included:

Using RAWMODE

HLLAPI RAWMODE allows the equivalent of datastream and high level access (similar to the OS/2 ULLAPI) for Windows. This is not a replacement for the IDK (which is still the only way to get total control over events under INFOConnect), but will allow for more precise control in your HLLAPI projects.

RAWMODE is one of two options available within the Set Session Parameters function that determines if raw host data will be received by the application. In RAWMODE, the appropriate functions look at raw data coming from the connection tool within Accessory Manager before data is delivered to the terminal tool.

The RAWMODE option affects the Copy Presentation Space to String and Copy String to Presentation Space functions.

The other option available within Set Session Parameters, PSMODE, is the default setting that allows applications to receive screen data.

SetSession Parameters Changes

Two new options, RAWMODE and PSMODE, have been added. The default setting would be PSMODE. In PSMODE, all HLLAPI functions would work identically to the way they do now (as documented). In RAWMODE, the appropriate functions would be looking at raw data coming from the connection tool within Accessory Manager before the data was delivered to the terminal tool. In fact, this would work like a hook procedure so that data wouldn’t be delivered to the terminal tool unless the HLLAPI application wanted to.

Accessory Manager Changes

When the frame is notified by the connection tool that data has been received, the frame has to determine if StartHostNotification has been turned on while in RAWMODE. If so, then it has to hold the data and notify HLLAPI of a host update. Then, once CopyPSToString has been called, copy the data into the buffer and discard it without delivering it to the emulator.

If CopyStringToPS is called with a PS Position of zero, then Accessory Manager will need to send the data directly to the connection tool. If CopyStringToPS is called with a PS Position of one, then the data must be delivered to the emulator as if it had just been received from the connection tool.

If StopHostNotification is called by the application, then Accessory Manager must release the communications data hook and allow all data from the connection tool to go directly to the terminal tool. While in RAWMODE, if CopyPSToString is called while host notification is disabled, then this function should always fail. However, CopyStringToPS can be called anytime after RAWMODE has been turned on. This would allow a HLLAPI application to simulate emulator transmits or to simulate host data.

Copy Presentation Space to String Function in RAWMODE

Because HLLAPI does not return the length of the buffer in wDataLength, but the application is required to know it, Accessory Manager stores communications data in the buffer, using the data length specified as a maximum length. The PS Position parameter is ignored. Upon return, the first two bytes of the data buffer will contain the length of the buffer.

CopyPSToString Changes

In RAWMODE, this function becomes the equivalent of a low-level communications receive function. The only changes would be that Accessory Manager would put communications data in the specified buffer using the data length specified as a maximum length. The PS position parameter would be ignored. Upon return, the first two bytes of the data buffer will contain the length of the buffer. This change is necessary because HLLAPI does not return the length of the buffer in wDataLength. The application is required to know the size of the buffer.

Copy String to Presentation Space Function in RAWMODE

The PS parameter setting determines how the data is transmitted to the terminal tool. It allows applications to modify host data before delivering it to the emulator.

PS Parameter Setting	Copy Sting to Presentation Space (15) Function
0	Transmits the data from the specified buffer (using the specified length) directly to the connection tool.
1	Transmits the data to the terminal tool as if the data had just been received.

CopyStringToPS Changes

In RAWMODE, this function will be similar to a low-level communications transmit function. If the PS parameter is set to zero, then this function will transmit the data from the specified buffer (using the specified length) directly to the connection tool. If the PS parameter is set to one, then the data specified will be delivered to the terminal tool as if the data had just been received. This provides the hook mechanism so that if a HLLAPI application doesn't want the emulator to see any host data, then the application just has to never call CopyStringToPS with a PS Position of one. This would also allow applications to modify host data before delivering it to the emulator.

Currently, when an application wants to do screen scraping through HLLAPI, the following pseudo-code shows the required steps:

```
StartHostNotification("A")

while
{
  ReturnCode = Pause() // Must have called SetSessionParameters with IPAUSE
  if ReturnCode = WHLLPSCHANGED
  {
    QueryHostUpdate()
    CopyPSToString()
  }
  // Once all of the appropriate data has been received, break out of the
  loop
}

StopHostNotification()
```

When the frame is notified by the connection tool that data has been received, the frame determines whether or not Start Host Notification (25) has been turned on while in RAWMODE. If so, the connection tool holds the data and notifies HLLAPI of a host update.

The following example in pseudo-code shows how an application could use the raw mode for host notification.

```
if (RawData) // Some value that determines whether the application wants raw
  data or screen data
  SetSessionParameters(“RAWMODE“)
else
  SetSessionParameters(“PSMODE“)


StartHostNotification(“A“)

while
{
  ReturnCode = Pause() // Must have called SetSessionParameters with IPAUSE
  if ReturnCode = WHLLPSCHANGED
  {
    QueryHostUpdate()
    if (RawData)
      CopyPSToString(Buffer,len) // This call will retrieve raw host data
      DataLen = *((short*)&Buffer[0]0;
      if (Transmit) // Some value that determines whether to transmit data
  to the host or deliver to the emulator
        CopyStringToPS(XmitBuffer, len, 0) // This transmits a buffer to the
  host bypassing the emulator
      else
        CopyStringToPS(&Buffer[sizeof(short)], DataLen, 1) // Sends the raw
  data to the emulator
    else
      CopyPSToString(Buffer, len) // This call will retrieve screen data
  }
  // Once all of the appropriate data has been received, break out of the
  loop
}

StopHostNotification()
```

ACMState F

In This Appendix This appendix explains the new utility ACMState.exe. The following sections are included:

Using ACMState.exe

ACMState.exe can be used to determine if a particular Accessory Manager UTS session is up and active, which means that the POLL indicator is displayed so that host communications are ready and you can proceed normally. The utility accepts three optional command line parameters:

timeout interval = A value indicated in seconds. (The default is 30 seconds.)

HLLAPI shortname = A single letter, in uppercase. (The default is A.)

Connection Status checking = (Optional) The string “/S” to request a Connection Status check. (The default is to omit it, and perform no Connection Status check).

This utility uses the Attachmate IHLLAPI interface, and it makes a call to the "ConnectPS" function. For example, if the given session (or the Accessory Manager itself) is not running, it is started. If the Accessory Manager and/or session are already running, then that will work too. Just be sure to start your other HLLAPI applications after ACMState has exited.

ACMState waits for the given session to start. If no Connection Status check is requested, it will return immediately once the UTS session is up. If Connection Status Checking is enabled, it immediately returns once the session reports a normal connection status. You can visually check the state, as ACMState outputs a line to indicate the HLLAPI and session state.

Possible states can be inspected programmatically via a return code. Those return code values are:

Value	Description
0	Success. The HLLAPI session is up (if no State Checking performed).
0	Success. The HLLAPI session is up and active (if State is Checked).
1	Timeout. Connection State is NOT normal and Timeout has expired.
2	Failed. There is a problem with the session or HLLAPI.
3	BadInput. The command line input was not recognized.

If you want to check the status of multiple UTS sessions, call the ACMState utility once for each session. By running the utility with an input timeout parameter of "0", ACMState will check the session's connection status only once and immediately report a result. In that case, it will not wait for a gateway or host connection to be established. This utility with work with any of the UTS transports (i.e., both PEPGate LAN Client and INT-1).

No changes to the product installation or configuration are needed. ACMState can be placed in any desired directory location. You will need to have IHLAPI32.DLL in the PATH. You can get a usage output by specifying a command line input of: ?, /?, -?, Help, /Help, or -Help. If the input is unrecognized, a usage statement will be displayed, and no HLLAPI calls will be made.

Time/Date stamps are now output at utility startup, after the "ConnectPS" is successfully completed, and (if Connection Status Checking is requested) when the utility returns a final status.

Glossary

application window	The window that includes the title bar, menu bar, toolbar, scroll bars, status bar, and display area for session windows.
character attributes	Attributes that determine how characters look on the screen. For example, blinking and underline are character attributes.
end-of-text (EOT)	A character used to indicate the end of the text.
ENQ	An abbreviation that appears on an T 27 session's status line and indicates that the session is in enquire mode. A session automatically enters this mode when the host is trying to send data and is unable to (for example, when the session is in local mode). You must put the session in receive mode to receive the data.
field attributes	Attributes that define the existence, appearance, or behavior of a field. For example, whether a field is protected or not is determined by the field attribute.
host	A mainframe, mini-computer, or information hub with which the PC communicates.

line transmission activity indicator (LTAI)	An abbreviation that appears on an T 27 session's status line and indicates that the session is connected to the host. If LTAI does not appear on the status line, a communication problem exists between the PC and the host.
operator information area (OIA)	A line of data that appears at the bottom of a session window (status line). The status line displays such information as which mode the session is in and the cursor location. The OIA is not considered part of the presentation space.
presentation space	A specific number of columns and rows that define the area within which the host or user can display or type data.
session	A named communication connection between a PC and a host. T27 and UTS support long file names, but all session files end in the .ADP extension.
session parameters	Options specified by the Set Session Parameters (9) function that determine how other IHLLAPI functions operate. For example, when the disconnect session parameter is set to DISCONLOG, the Disconnect Presentation Space (2) function only disconnects your application from the session. When this parameter is set to DISCONPHYS, this function disconnects your application from the session, disconnects the session from the host, and closes the session.
session window	A window within Accessory Manager's application window that displays communication between the PC and the host.
short name	A one-character (A–Z), case-insensitive name used by IHLLAPI applications to identify a particular session. You can associate short names with session file names via Accessory Manager's Global Preferences dialog box or programmatically via the Associate Profile (911) function.
terminal keystroke	A keystroke that a terminal sends to a host. Terminal keystrokes (such as Transmit) typically do not exist on a PC keyboard and must be emulated using various keystroke combinations.

Index