WD_API Language Reference

Purpose

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

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

Introduction

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

The Wall Data application programming interface (WD_API) is one such API. Introduced originally by Wall Data Corporation in the middle 1990s for that company's RUMBA emulator product, WD_API was a proprietary implementation of the already-popular programming interface, EHLLAPI, employed successfully throughout business and industry for a wide range of automation tasks.

Accessing Attachmate 32-bit WD_API

In brief, an application accesses this interface by

1. ensuring Attachmate software, including dynamic load library EHLAPI32.DLL, is in the system search path, so it will be found and loaded when referenced.

2. declaring in application code specific reference to required WD_API entry points and associated parameter lists. Such references will depend on the application programming language used; the declaration below, for Visual Basic, is typical.

```
Declare Function WD_ConnectPS Lib "EHLAPI32.DLL" _
(ByVal hInstance As Long, ByVal ShortName As String) As Long
```

3. invoking WD_API functions as needed in application code.

Language support files used by WD_API (.h, .lib, .bas, and so on) are not distributed by Attachmate, so the information here is really only useful for those who are supporting existing WD_API applications. Header and lib files for EHLLAPI, WinHLLAPI, and Attachmate HLLAPI are distributed with EXTRA!.

WD_API Functions

Entry point

- WD_ConnectPS
- WD_DisconnectPS
- WD_SendKey
- WD_Wait
- WD_CopyPS
- WD_SearchPS
- WD_QueryCursorLocation
- WD_CopyPSToString
- WD_SetSessionParamEx
- WD_QuerySessions
- WD_Reserve
- WD_Release
- WD_CopyOIA
- WD_QueryFieldAttribute
- WD_CopyStringToPS
- WD_Pause
- WD_QuerySystem
- WD_ResetSystem
- WD_QuerySessionStatus
- WD_StartHostNotification
- WD_QueryHostUpdate
- WD_StopHostNotification
- WD_SearchField
- WD_FindFieldPosition
- WD_FindFieldLength
- WD_CopyStringToField
- WD_CopyFieldToString
- WD_DeletePS
- WD_SetCursor
- WD_StartKSIntercept
- WD_GetKey
- WD_PostInterceptStatus
- WD_StopKSIntercept
- WD_SendFile
- WD_ReceiveFile
- WD_Convert
- WD_ConnectWindowServices
- WD_DisconnectWindowServices
- WD_QueryWindowCoordinates
- WD_WindowStatus
- WD_ChangeWindowName
- WD_RunProfile
- WD_ShowSession

What information is provided for each function?

For each WD_API function, the following information is presented:

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

Prerequisites

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

Applicable session parameters

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

Call parameters

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

Return parameters

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

Notes or tips

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

WD_ConnectPS

This function connects a WD_API application to a specified presentation space (PS). If the application already has a connection, the connected PS is automatically disconnected, and a new connection established. An exclusive connection is established with WD_API between the client application program and the PS that requires the target session to be defined in the current EXTRA! configuration. An application program must call this function before requesting any of the following-listed functions.

Function

- WD_DisconnectPS
- WD_SendKey
- WD_Wait
- WD_CopyPS
- WD_SearchPS
- WD_QueryCursorLocation
- WD_CopyPSToString
- WD_Reserve
- WD_Release
- WD_CopyOIA
- WD_QueryFieldAttribute
- WD_CopyStringToPS
- WD_SearchField
- WD_FindFieldPosition
- WD_FindFieldLength
- WD_CopyStringToField
- WD_CopyFieldToString
- WD_SetCursor

Syntax

```
DWORD WD_ConnectPS(HWND hwnd, LPSTR szDataStr);
```

Prerequisites

Target sessions must be defined in the current EXTRA! configuration.

Applicable session parameters

The following session parameters from Function 9 affect this function.

WRITE_SUPER (default)
This application requires write access and allows only supervisory applications to connect to its PS.

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

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

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

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

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

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

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

Call parameters

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

Function	WD_ConnectPS
hwnd	Window handle of the application
szDataStr	1-character session short name; which must be a letter of the alphabet (A–Z).

Return parameters

Result code

Function returns one of the following codes:

Code	Description
0	The function was successful; the host presentation space is unlocked and ready for input.
1	An invalid host presentation space ID was entered.
4	Connection succeeded, but the host PS was busy.
5	Connection succeeded, but the host PS was locked (input inhibited).
9	A system error occurred.
11	The requested PS was in use by another application.

Example

```
/* Connect with session B */
DWORD Result = WD_ConnectPS(hWnd, "B");
```

Hint

If the EXTRA! session specified has not already been started when this function is called, calling this function will start the session in hidden state. Because function 1 returns immediately, the result code will be 5 (PS locked). Before attempting to use the session, the application should repeatedly call function 4, “Wait,” until a 0 (Success) result code is obtained.

WD_DisconnectPS

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

An application automatically disconnects from the currently connected PS when it connects to another PS.

A WD_API application program should call this function to disconnect from the currently connected PS before exiting.

Syntax

```
DWORD WD_DisconnectPS(HWND hwnd);
```

Prerequisites

WD_ConnectPS

Applicable session parameters

The following session parameter from Function 9 affects this function.

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

Call parameters

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

Function	WD_DisconnectPS
hwnd	Window handle of the application

Return parameters

Result code

Function returns one of the following codes:

Code	Description
0	The function was successful.
1	The application was not connected with a host PS.
9	A system error occurred.

Example

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

Hint

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

WD_SendKey

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

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

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

Syntax

```
DWORD WD_SendKey(HWND hwnd, LPSTR szDataStr);
```

Prerequisites

WD_ConnectPS

The keyboard must be unlocked before keystrokes will be accepted.

Applicable session parameters

The following session parameters from Function 9 affect this function.

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

AUTORESET (default)
Attempts to reset inhibited conditions by adding the RESET prefix to all keystroke strings sent.
"
NORESET
Does not add RESET prefix to key strings.

Call parameters

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

Function	WD_SendKey
Hwnd	Window handle of the application
Data string	A string of maximum 128 characters (keystrokes)to be sent to the host PS, terminated with a null character.

Return parameters

Result code

Function returns one of the following codes:

Code	Description
0	The function was successful.
1	The application was not connected with a host PS.
2	An incorrect parameter was entered.
4	Host session was busy; not all keystrokes were sent.
5	Host session was inhibited, not all keystrokes were sent.
9	A system error occurred.

Example

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

Tips

- For increased performance, an application may send entire strings using Function 33, “Copy String to Field,” or Function 15, “Copy String to Presentation Space,” rather than using this function; however, only function 3 may send special control keys.
- If the keystroke string is longer than 128 characters (which is the Send Key function’s limit), use multiple calls to the Send Key function.

WD_Wait

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

The Wait function is not a good method for determining when the host is ready for input. This function is provided to determine if the terminal session can accept keystrokes (using "Send Key" or a copy function). To determine when the host is ready, the application should search the screen for key fields, usually near the bottom of the screen. Another method is to query the cursor position until it is located at the correct field. Because host applications are so different and a terminal cannot determine when a host application is ready for input, the WD_API application should determine when the host is ready for more input.

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

Syntax

```
DWORD WD_Wait(HWND hwnd);
```

Prerequisites

WD_ConnectPS

Applicable session parameters

The following session parameters from Function 9 affect this function.

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

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

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

Call parameters

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

Function	WD_Wait
Hwnd	Window handle of the application

Return parameters

Result code

Function returns one of the following codes:

Code	Description
0	The function was successful; host PS is unlocked and ready for input.
1	The application was not connected with a host PS.
4	Function timed out while in XCLOCK or XSTATUS state.
5	Keyboard is locked.
9	A system error occurred.

Example

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

Tips

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

WD_CopyPS

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

Syntax

```
DWORD WD_CopyPS(HWND hwnd, LPSTR szDataStr, WORD wLen);
```

Prerequisites

WD_ConnectPS

Applicable session parameters

The following session parameters from Function 9 affect this function.

NOATTRB (default)
Attribute bytes and other characters not displayable in ASCII are translated into blanks.

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

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

NOEAB (default)
EABs are not copied.

XLATE
EABs are translated to CGA text mode attributes.

NOXLATE (default)
EABs are not translated.

Call parameters

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

Function	WD_CopyPS
hwnd	Window handle of the application
szDataStr	A string large enough to accommodate data from the current PS display Model (including EABs if requested). See chart below.
WLen	Not applicable (PS length implied; see chart).

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

Return parameters

Data string

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

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

Result code

Function returns one of the following codes:

Code	Description
0	Success; text from the PS has been copied to data string.
1	The application was not connected with a host PS.
4	The copy was successful, but PS was waiting for host response.
5	The copy was successful, but the keyboard is locked.
9	A system error occurred.

Example

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

Tips

- Use this function only when the entire PS is needed; otherwise, use Function 8, “Copy Presentation Space to String,” or Function 34, “Copy Field to String.”
- Use Function 10, “Query Sessions,” or Function 22, “Query Session Status,” to check host session PS size (which may be changed by the host).
- This function does not format the data string returned. To format the string for printing and have the information appear as it does in EXTRA!, the application must determine the number of columns currently displayed (use function 22, “Query Session Status,” for this purpose), then insert a line break (newline, or CR LF) at the end of each line (that is, that many columns).

WD_SearchPS

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

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

Syntax

```
DWORD WD_SearchPS (HWND hwnd, LPWORD lpwFound, WORD wPSP,
                                LPSTR szSearchStr);
```

Prerequisites

WD_ConnectPS

Applicable session parameters

The following session parameters from Function 9 affect this function.

SRCHALL and SRCHFRWD (default)
The function scans the entire PS for the first occurrence of the specified string.

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

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

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

Call parameters

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

Function	WD_SearchPS
hwnd	Window handle of the application
lpwLocation	Reserved for location where found (integer)
wPSP	Start position where the search function is to begin (SRCHFRWD) or to end (SRCHBKWD). This parameter is ignored if SRCHALL is set.
szSearchStr	Text to be searched for in the PS

Return parameters

PS Position

Function replaces the value of call parameter Location with the PS position where specified text was found, or 0 if the text was not found.

Result code

Function returns one of the following codes:

Code	Description
0	The function was successful (the specified text was found).
1	The application was not connected with a host PS.
2	An incorrect parameter was entered.
7	An invalid PS position was specified for beginning the search
9	A system error occurred
24	The specified text was not found.

Example

```
WORD Location;
/* Search for: "Hello" starting at PS position 199 */
DWORD Result = WD_SearchPS(hWnd, &Location, 199, "Hello");
```

Tips

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

WD_QueryCursorLocation

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

Syntax

```
DWORD WD_QueryCursorLocation(HWND hwnd, LPWORD lpwLocation);
```

Prerequisites

WD_ConnectPS

Applicable session parameters

None.

Call parameters

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

Function	WD_QueryCursorLocation
hwnd	Window handle of the application
lpwLocation	Reserved for location where found

Return parameters

PS Position

Function replaces the value of call parameter Location with the PS position of the cursor.

Result code

Function returns one of the following codes:

Code	Description
0	The function was successful (the specified text was found).
1	The application was not connected with a host PS.
9	A system error occurred

Example

```
WORD Location;
DWORD Result = WD_QueryCursorLocation(hWnd, &Location);
```

Tips

- This function is one method of determining whether a host session is at a particular screen, assuming the position where the cursor will appear on that screen is known in advance.
- To make this determination, the application can repeatedly query cursor position until it is located at the correct field.

WD_CopyPSToString

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

Syntax

```
DWORD WD_CopyPSToString(HWND hwnd, WORD wPSP,
                        LPSTR szDataStr, WORD wLen);
```

Prerequisites

WD_ConnectPS

Applicable session parameters

The following session parameters from Function 9 affect this function.

NOATTRB (default) Attribute bytes and other characters not displayable in ASCII are translated into blanks. ATTRB Attribute bytes and other characters not displayable in ASCII are not translated.

EAB Extended Attribute Bytes are copied. Two characters are placed in the application data string for each one that appears in the PS. The EAB is the second character. To accommodate this, the application program must allocate a data string that is twice the number of displayable characters to be copied. For example, 160 bytes should be allotted to copy the first 80 characters with EABs. NOEAB (default) Extended Attribute Bytes are not copied.

XLATE Extended Attribute Bytes are translated to CGA text mode attributes. NOXLATE (default) Extended Attribute Bytes are not translated.

Call parameters

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

Function	WD_CopyPSToString
hwnd	Window handle of the application
wPSP	The PS position where the copying should begin.
szDataStr	A string of sufficient size to hold data requested from the PS, including EABs if requested
wLen	The number of characters allocated in Data string.

Return parameters

Data string

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

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

Result code

Function returns one of the following codes:

Code	Description
0	The function was successful; requested data was copied to the string.
1	The application program was not connected to a valid PS.
2	String length was specified as zero, or extended past the end of the PS.
4	Requested data was copied, but the PS was waiting for host response.
5	Requested data was copied, but the keyboard was locked.
7	An invalid PS position was specified for beginning the copy.
9	A system error occurred.

Example

```
/* Start position to copy */
WORD PsPos = 199;
/* Buffer for returned data */
char DataStr[6];
/* Length of string to copy */
WORD DataLn = 5;
DWORD Result = WD_CopyPSToString(hWnd, PsPos, DataStr, DataLn);
```

WD_SetSessionParamEx

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

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

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

Syntax

```
DWORD WD_SetSessionParamEx(HWND hwnd, LPSTR szDataStr);
```

Prerequisites

None.

Applicable session parameters

None.

Call parameters

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

Function	WD_SetSessionParamEx
hwnd	Window handle of the application
szDataStr	A string containing one or more session parameters, which can be separated by commas or blanks. Valid session parameters are described below.

Copy parameters

The following session parameters affect all copy functions.

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

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

EAB
Extended Attribute Bytes are copied along with data.

NOEAB (default)
EABs are not copied (data only).

XLATE
Copied Extended Attribute Bytes are translated to CGA color codes.

NOXLATE (default)
Copied Extended Attribute Bytes are returned without translation.

Connect parameters

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

CONLOG (default)
When Function 1, “Connect Presentation Space,” is called, the emulator session corresponding to the target PS does **not** become the active application. The calling application remains active. Likewise, when Function 2, “Disconnect Presentation Space,” is called, the calling application remains active.

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

WRITE_SUPER (default)
This parameter is set by a client application program that requires write access and allows only supervisory applications to connect to its PS.

WRITE_WRITE
This parameter is set by a client application program that requires write access and allows other applications that have predictable behavior to connect to its PS.

WRITE_READ
This parameter is set by a client application program that requires write access and allows other applications to use read-only functions on its PS.

WRITE_NONE
This parameter is set by a client application program that requires exclusive access to its PS. No other applications will have access to its PS.

SUPER_WRITE
This parameter is set by a supervisory client application program that allows applications with write access to share the connected PS. The client application program setting this parameter will not cause errors for other applications, but will provide only supervisory-type functions.

WRITE_READ
This parameter is set by a client application program that requires read-only access and allows other applications that perform read-only functions to connect to its PS.

Esc/Reset parameters

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

ESC= char
Specifies the escape character for keystroke mnemonics (“@” is the default). Blank is not a valid escape value.

AUTORESET (default)
Attempts to reset all inhibited conditions by adding the prefix RESET to all keystroke strings sent using Function 3, “Send Key.
”

NORESET
Does not add RESET prefix to function 3 key strings.

Search parameters

The following session parameters affect all search functions.

SRCHALL (default)
Scans the entire PS or field.

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

SCRCHFRWD (default)
Performs the scan in an ascending direction.

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

Wait parameters

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

TWAIT (default)
For Function 4, "Wait," TWAIT waits up to a minute before timing out on XCLOCK or XSYSTEM.
For Function 51, "Get Key," TWAIT does not return control to the WD_API client application program until it has intercepted a key (a normal or AID key, based on the option code specified under Function 50, "Start Keystroke Intercept").

LWAIT
For Function 4, "Wait," LWAIT waits until XCLOCK / XSYSTEM clears. This option is not recommended because XSYSTEM or permanent XCLOCK will prevent control being returned to the application.
For Function 51, "Get Key," LWAIT does not return control to your application until it has intercepted a key. The intercepted key could be a normal or AID key, based on the option specified under Function 50, "Start Keystroke Intercept."

NWAIT
For Function 4, "Wait," NWAIT checks status and returns immediately (no wait).
For Function 51, "Get Key," NWAIT returns code 25 (keystroke not available) if nothing matching the option specified under Function 50, "Start Keystroke Intercept," is queued.

Pause parameters

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

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

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

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

Transfer parameters

The following session parameters affect Function 90, “Send File,” and Function 91, “Receive File.”

NOQUIET (default) Displays SEND and RECEIVE messages showing progress of the file transfer. QUIET Does not display SEND and RECEIVE messages.

Return parameters

Parameters accepted

Function replaces the value of call parameter Data length with the number of session parameters that were set.

Result code

Function returns one of the following codes:

Code	Description
0	The function was successful.
2	One or more parameter names were not recognized; all recognized parameters were accepted.
9	A system error occurred.

Example

```
/* Set session parameters */
DWORD Result = WD_SetSessionParamEx(hWnd, "SRCHFROM,SRCHFRWD");
```

WD_QuerySessions

This function returns summary information about each currently started session. The information is returned in a 12-byte data string for each session.

Syntax

```
DWORD WD_QuerySessions(HWND hwnd, LPWORD lpwCount,
                       LPQUERYSESSION lpQS);
```

Prerequisites

None.

Applicable session parameters

The following session parameters from Function 9 affect this function.

NOCFGSIZE The function returns the current size of the connected PS. CFGSIZE (default) The function returns the configured size of the PS, ignoring any host overrides

.

Call parameters

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

Function	WD_QuerySessions
hwnd	Window handle of the application
lpwCount	number of sessions to query about
lpQS	Address of an array of `QUERYSESSION` structures

Return parameters

Sessions started

Function replaces the value of call parameter Sessions count with the number of started sessions for which information was returned.

Session information

Function replaces content of call parameter Data string with information about currently-open sessions, twelve bytes per session, as follows:

Byte	Description
1	Session short name.
2–9	Session long name.
10	A fixed constant byte value (‘H’).
11–12	PS size in binary.

Result code

Function returns one of the following codes:

Code	Description
0	The function was successful.
2	String length specified was not valid.
9	A system error occurred.

Example

```
WORD Sesscount = 26;
/* 12 bytes per session, max. 26 sessions */
QSSTRUCT qs[26];
DWORD Result = WD_QuerySessions(hWnd, &Sesscount, &qs);
```

Tip

Text returned as the session "long name" will be the first eight characters of the name of the configuration ("*.EDP") file used to open the session:

WD_Reserve

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

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

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

Syntax

```
DWORD WD_Reserve(HWND hwnd);
```

Prerequisites

WD_ConnectPS

Applicable session parameters

None.

Call parameters

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

Function	WD_Reserve
hwnd	Window handle of the application

Return parameters

Result code

Function returns one of the following codes:

Code	Description
0	The function was successful.
1	The application is not connected to a valid PS.
5	Presentation space cannot be used.
9	A system error occurred.

Example

```
DWORD Resullt = WD_Reserve(hWnd);
```

WD_Release

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

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

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

Syntax

```
DWORD WD_Release(HWND hwnd);
```

Prerequisites

WD_ConnectPS

Applicable session parameters

None.

Call parameters

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

Function	WD_Release
hwnd	Window handle of the application

Return parameters

Result code

Function returns one of the following codes:

Code	Description
0	The function was successful.
1	The application is not connected to a valid PS.
9	A system error occurred.

Example

```
DWORD Resullt = WD_Release(hWnd);
```

WD_CopyOIA

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

Syntax

```
DWORD WD_CopyOIA(HWND hwnd, LPSTR szDataStr, WORD wLen);
```

Prerequisites

WD_ConnectPS

Applicable session parameters

None.

Call parameters

The following session parameters from Function 9 affect this function.

Function	WD_CopyOIA
hwnd	Window handle of the application
szDataStr	A pre-allocated 103-byte data string
wLen	103

Return parameters

OIA data

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

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

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

Result code

Function returns one of the following codes:

Code	Description
0	OIA data were copied; PS is unlocked.
1	The application is not connected to a valid PS.
2	Data string length specified was not valid.
4	OIA data were copied, but the PS is busy.
5	OIA data were copied, but the keyboard is locked.
9	A system error occurred.
10	Copy OIA is not supported on UNIX/VAX hosts.

Example

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

WD_QueryFieldAttribute

This function returns the field attribute byte for the PS position.

The returning parameter contains the field attribute for the specified PS position. The value of the attribute byte is C0-DF (unprotected field attributes) and E0-FF (protected attributes). A zero attribute means that no field attribute was found in the PS.

Syntax

```
DWORD WD_QueryFieldAttribute(HWND hwnd, LPWORD lpwAtt,
                             WORD wPSP);
```

Prerequisites

WD_ConnectPS

Applicable session parameters

None.

Call parameters

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

Function	WD_QueryFieldAttribute
hwnd	Window handle of the application
lpwAtt	Receives field attribute value.
wPSP	The PS position for which field information is wanted

.

Return parameters

Attribute value

Function replaces the value of call parameter Attribute with the attribute byte for the specified field. If zero, the PS is unformatted and no attribute can be returned.

3270 Field attribute

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

Attribute value, continued

5250 Field attribute

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

Result code

Function returns one of the following codes:

Code	Description
0	Function was successful.
1	The application is not connected to a valid PS.
7	An invalid PS position was specified.
9	A system error occurred.
10	Query Field Attribute is not supported on UNIX/VAX hosts.
24	The PS was unformatted.

Example

```
WORD Attribute;
/* Query field attribute at position 199 */
DWORD Result = WD_QueryFieldAttribute(hWnd, &Attribute, 199);
```

WD_CopyStringToPS

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

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

Syntax

```
DWORD WD_CopyStringToPS(HWND hwnd, WORD wPSP,
                                LPSTR szDataStr, WORD wLen);
```

Prerequisites

WD_ConnectPS

Applicable session parameters

The following session parameters from Function 9 affect this function.

EAB Extended Attribute Bytes are copied. Two characters are copied from the application data string for each position in the PS. The EAB is the second character. NOEAB (default) Extended Attribute Bytes are not present.

XLATE Extended Attribute Bytes are translated from CGA text mode attributes. NOXLATE (default) Extended Attribute Bytes are not translated.

Call parameters

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

Function	WD_CopyStringToPS
hwnd	Window handle of the application
wPSP	Position of the PS where function is to begin copying data.
szDataStr	ASCII text and, if requested, EABs to be copied into the PS.

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

wLen	Data string length.

Note: This function cannot send keyboard mnemonics.

Return parameters

Result code

Function returns one of the following codes:

Code	Description
0	Function was successful.
1	The application is not connected to a valid PS.
2	Function was called with an invalid parameter.
5	PS is busy or locked, or the data string contained illegal data. The string was not copied.
6	The string was copied, but truncated at the end of the field or screen.
7	An invalid PS position was specified.
9	A system error occurred.
10	Copy String to Presentation Space not supported on UNIX/VAX hosts.

Example

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

Tip

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

Result code 6 indicates attempt was made to copy data into a protected field. Before writing application code to use this function, the programmer should check that the location where data are to be copied to the PS is (a) an unprotected field and (b) of sufficient extent to accept all the data to be copied.

WD_Pause

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

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

Syntax

```
DWORD WD_Pause(HWND hwnd, WORD wTime);
```

Prerequisites

WD_StartHostNotification must be called if the application program uses session parameter IPAUSE."

Applicable session parameters

The following session parameters from Function 9 affect this function.

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

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

Call parameters

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

Function WD_Pause

hwnd Instance handle of the application

wTime The pause duration in 1/2-second multiples.

Return parameters

Result code

Function returns one of the following codes:

Code	Description
0	The pause duration expired.
9	A system error occurred.
26	A host session update of the requested type has occurred.

Example

```
/* Wait for 10 sec */
DWORD Result = WD_Pause(hWnd, 20);
```

WD_QuerySystem

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

Syntax
```
DWORD WD_QuerySystem(HWND hwnd, PQSYSSTRUCT pQSS);
```

Prerequisites
None.

Applicable session parameters
None.

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

Function	WD_QuerySystem
hwnd	Window handle of the application
pQSS	Address of a QSYSSTRUCT object.

Return parameters

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

Byte	Description
1	Version number.
2–3	Level number.
4–9	Date (month, date, year).
10–12	NA
13	Always “U” (unable to determine)
14–18	NA
19	Reserved
20-23	Extended error code 1 (system component, printable ASCII)
24-27	Always zero
28-35	NA

Result code
Function returns one of the following codes:

Code	Description
0	The function was successful; the data string was returned.
9	A system error occurred.

Example
```
QSYSSTRUCT qs;
DWORD Result = WD_QuerySystem(hWnd, &qs);
```

WD_ResetSystem

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

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

Syntax

```
DWORD WD_ResetSystem(HWND hwnd);
```

Prerequisites

None.

Applicable session parameters

None.

Call parameters

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

Function	WD_ResetSystem
hwnd	Window handle of the application

Return parameters

Result code

Function returns one of the following codes:

Code	Description
0	The function was successful; the data string was returned.
9	A system error occurred.

Example

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

Tip

If a session is visible when this function is called, the session will not be released from memory, though any WD_API connection of the application with the session will be disconnected.

WD_QuerySessionStatus

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

- Short and long names
- Terminal type
- Number of rows and columns in the PS

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

Syntax

```
DWORD WD_QuerySessionStatus(HWND hwnd, PQSSSTRUCT pQS);
```

Prerequisites

None.

Applicable session parameters

None.

Call parameters

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

Function	WD_QuerySessionStatus
hwnd	Window handle of the application
pQS	Address of a QSSTATUS object whose first byte is: • a session short name • blank or null, indicating the currently-connected PS, or • asterisk ('*'), indicating the session currently with keyboard focus

Return parameters

Session information

Function replaces content of call parameter Data string with information about the session, organized as follows:

Byte	Description
1	Session short name or blank or null.
2-9	Session long name.
10	Session type: ‘D’ = 3270 Host ‘F’ = 5250 Host ‘V’ = UNIX or VAX
11	Session characteristics: Bit 0: 0=No EAB; 1=EABs Bit 1: 0=No programmed symbols 1=Programmed symbols Bit 2–7: Reserved
12–13	Number of rows (binary).
14–15	Number of columns (binary).
16–18	Reserved

Result code

Function returns one of the following codes:

Code	Description
0	The function was successful; the data string was returned.
1	An invalid session short name was specified.
2	An invalid string length was sent to the function.
9	A system error occurred.

Example

```
QSSSTRUCT qs;
/* Request status of session with keyboard focus */
qs.Shortname = '*';
int Result = WD_QuerySessionStatus(hWnd, &qs);
```

WD_StartHostNotification

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

Syntax

```
DWORD WD_StartHostNotification(HWND hwnd, PSHNSTRUCT pSHN);
```

Prerequisites

None.

Applicable session parameters

None.

Call parameters

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

Function	WD_StartHostNotification
hwnd	Window handle of the application
pSHN	Address of a `SHNSTRUCT` object.

Data string format

Byte	Description
1	Session short name. If blank or null, the session to which the application is currently connected.
2	One of the following characters: "P"—Notification of PS update "O"—Notification of OIA update "B"—Notification of both OIA and PS updates
3-6	Reserved

Return parameters

Result code

Function returns one of the following codes:

Code	Description
0	The function was successful.
1	An invalid session short name was specified.
2	An invalid parameter was specified.
9	A system error occurred.

Example

```
SHNSTRUCT shn;
/* Short name of session */
shn.Shortname = 'E';
/* Both OIA and PS updates */
shn.Event = 'B';
int Result = WD_StartHostNotification(hWnd, &shn);
```

WD_QueryHostUpdate

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

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

Syntax

DWORD WD_QueryHostUpdate(HWND hwnd, LPSTR szPSID);

Prerequisites

WD_StartHostNotification

Applicable session parameters

None.

Call parameters

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

Function	WD_QueryHostUpdate
hwnd	Window handle of the application
szPSID	A one-byte string containing the short name of the desired session, or blank or null requesting the connected session.

Return parameters

Result code

Function returns one of the following codes:

Code	Description
0	No updates occurred since the last call.
1	An invalid PS was specified.
8	Function 23, “Start Host Notification,” has not been called for this PS.
9	A system error occurred.
21	The OIA was updated.
22	The PS was updated.
23	Both OIA and PS were updated.

Example

```
DWORD Result = WD_QueryHostUpdate(hWnd, "B");
```

WD_StopHostNotification

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

Syntax

```
DWORD WD_StopHostNotification(HWND hwnd, LPSTR szPSID);
```

Prerequisites

WD_StartHostNotification

Applicable session parameters

None.

Call parameters

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

Function	WD_StopHostNotification
hWnd	Instance handle of the application
szPSID	A one-byte string containing the short name of the desired session, or blank or null requesting the connected session.

Return parameters

Result code

Function returns one of the following codes:

Code	Description
0	The stop notification was successful.
1	An invalid session short name was specified.
8	Function 23, “Start Host Notification,” has not been called for this PS.
9	A system error occurred.

Example

```
DWORD Result = WD_StopHostNotification(hWnd, "B");
```

WD_SearchField

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

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

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

Syntax

```
DWORD WD_SearchField(HWND hwnd, LPWORD lpwLocation,
                                    WORD wPSP, LPSTR szSearchText);
```

Prerequisites

WD_ConnectPS

Applicable session parameters

The following session parameters from Function 9 affect this function.

SRCHALL (default) The entire field containing the specified PS position is searched. SRCHFROM Search begins at the specified position in the field.

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

Call parameters

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

Function	WD_SearchField
hwnd	Window handle of the application
lpwLocation	Receive location value if text is found
wPSP	Specifies a PS position within the target field or on the field attribute that begins it.. For SRCHALL, this can be any PS position within the field. For SRCHFROM, search begins here for SRCHFRWD or ends here for SRCHBKWD.
szSearchString	ASCII text to be searched for in the field

Return parameters

PS Position

Function replaces the value of call parameter Location with the PS position where the specified text was found. If zero, the specified text was not found.

Result code

Function returns one of the following codes:

Code	Description
0	The function was successful; the string was found.
1	The application is not connected to a valid PS.
2	The string length was zero; or, if STREOT was in effect, no EOT character was found in the given search string.
7	An invalid PS position was specified.
9	A system error occurred.
10	Search Field is not supported on UNIX or VAX hosts.
24	The string was not found.

Example

```
WORD Location;
/* Search field at PS position 199 for "Hello" */
DWORD Result = WD_SearchField(hWnd, &Location, 199, "Hello");
```

WD_FindFieldPosition

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

Syntax
DWORD WD_FindFieldPosition(HWND hwnd, LPWORD lpwLocation,
WORD wPSP, LPSTR szCode);

Prerequisites
WD_ConnectPS

Applicable session parameters
None.

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

Function	WD_FindFieldPosition
hwnd	Window handle of the application
lpwLocation	Receives location of field
PS position	Specifies a PS position within the field or on the field attribute that begins it.
szCode	A 2-character code specifying the field to find. (See format below.)

Data string format

Content	Description
^^ or T^	This field.
N^	Next field (protected or unprotected).
NP	Next protected field.
NU	Next unprotected field.
P^	Previous field (protected or unprotected)
PP	Previous protected field.
PU	Previous unprotected field.

^ = a space

Return parameters

Field position
Function replaces the value of call parameter Location with the PS position where the specified field begins. If zero, the field is either zero length or the PS is unformatted.

Result code
Function returns one of the following codes:

Code	Description
0	The function was successful; the field was found.
1	The application is not connected to a valid PS.
2	An incorrect parameter was specified.
7	An invalid PS position was specified.

9	A system error occurred.
24	Either the field was not found, or the PS was unformatted.
28	The field length is zero bytes.

Example

```
WORD  PSLoc;
/* Find start of this field (t, space) beginning at 199*/
DWORD Result = WD_FindFieldPosition(hWnd, &PSLoc, 199, "T ");
```

WD_FindFieldLength

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

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

Syntax

```
DWORD WD_FindFieldLength(HWND hwnd, LPWORD lpwLength,
                         WORD wPSP, LPSTR szCode);
```

Prerequisites

WD_ConnectPS

Applicable session parameters

None.

Call parameters

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

Function	WD_FindFieldLength
hwnd	Window handle of the application
lpwLength	Receives length of found field
wPSP	Specifies a PS position within the field or on the field attribute that begins it.
szCode	A 2-character code specifying the field to find. (See format below.)

Data string format

Content	Description
^^ or T^	This field.
N^	Next field (protected or unprotected).
NP	Next protected field.
NU	Next unprotected field.
P^	Previous field (protected or unprotected)
PP	Previous protected field.
PU	Previous unprotected field.

^ = a space

Return parameters

Field length

Function replaces the value of call parameter Length with the length of the specified field. If zero, the field was not found, or is zero length, or the PS is unformatted.

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

Result code

Function returns one of the following codes:

Code	Description
0	The function was successful; the field was found.
1	The application is not connected to a valid PS.
2	An incorrect parameter was specified.
7	An invalid PS position was specified.
9	A system error occurred.
24	Either the field was not found, or the PS was unformatted.

Example

```
WORD Length;
/* Find length of this field (t, space) beginning at 199*/
DWORD Result = WD_FindFieldLength(this.hWnd, &Length, 199, "T ");
```

WD_CopyStringToField

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

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

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

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

Syntax

```
DWORD WD_CopyStringToField(HWND hwnd, WORD wPSP, LPSTR szData);
```

Prerequisites

WD_ConnectPS

Applicable session parameters

The following session parameters from Function 9 affect this function.

EAB Text and EABs are copied from the data string. NOEAB (default) The data string does not contain EABs.

XLATE EABs are translated from CGA text mode attributes. NOXLATE (default) EABs are copied without translation.

Call parameters

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

Function	WD_CopyStringToField
hwnd	Window handle of the application
wPSP	A position in the PS that lies within the field or on the field attribute that begins it. Copy always starts at the beginning of the field.
szData	ASCII text to be copied into the field.

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

Return parameters

Result code

Function returns one of the following codes:

Code	Description
0	Success; the string was copied to the target field in the PS.
1	The application is not connected to a valid PS.
2	A string length of zero was specified.
5	Either the target field was protected or inhibited; or a nondisplayable character was included in the string.
6	The string was copied, but it was truncated because the field was shorter than the string.
7	An invalid PS position was specified
9	A system error occurred
10	Copy String to Field is not supported on UNIX and VAX hosts
24	The host PS is unformatted

Example

```
/* Copy "Hello World" to field containing position 199 */
DWORD Result = WD_CopyStringToField(hWnd, 199, "Hello World");
```

WD_CopyFieldToString

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

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

This function ends when one of two conditions is met:

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

Syntax

```
DWORD WD_CopyFieldToString(HWND hwnd, WORD wPSP,
                           LPSTR szData, WORD wLen);
```

Prerequisites

WD_ConnectPS

Applicable session parameters

The following session parameters from Function 9 affect this function.

EAB Text and EABs are copied to the buffer. NOEAB (default) EABs are not copied.

XLATE EABs are translated to CGA text mode attributes. NOXLATE (default) EABs are not translated.

Call parameters

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

Function	WD_CopyFieldToString
hwnd	Window handle of the application
wPSP	A position in the PS that lies within the field or on the field attribute that begins it. Copy starts at the beginning of the field.
szData	The string to which the program copies the contents of the field.
wLen	The number of characters to be copied.

NOTE: Data string must be at least twice this length if the EAB session parameter is set.

Return parameters

Field content

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

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

Result code

Function returns one of the following codes:

Code	Description
0	Success; text from the field has been copied to data string.
1	The application was not connected with a host PS.
2	A parameter error was detected.
6	The string was copied, but it was truncated because the string was shorter than the field.
7	An invalid PS position was specified.
9	A system error occurred.
10	Copy Field to String not supported on UNIX and VAX hosts
24	The presentation space is unformatted.

Example

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

WD_DeletePS

This function disconnects the specified session and closes it.

Syntax

```
DWORD WD_DeletePS(HWND hwnd, LPSTR szPSID);
```

Prerequisites

WD_ConnectPS

Applicable session parameters

None.

Call parameters

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

Function	WD_DeletePS
hwnd	Window handle of the application
szPSID	A 1-byte string containing the session short name.

Return parameters

Result code

Function returns one of the following codes:

Code	Description
0	Success; text from the field has been copied to data string.
1	The application was not connected with a host PS.
9	A system error occurred.

Example

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

WD_SetCursor

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

Prerequisites

WD_ConnectPS

Syntax

```
DWORD WD_SetCursor(HWND hwnd, WORD wLocation);
```

Applicable session parameters

None.

Call parameters

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

Function	WD_SetCursor
hwnd	Window handle of the application
wLocation	The desired cursor position in the PS.

Return parameters

Result code

Function returns one of the following codes:

Code	Description
0	Success; text from the field has been copied to data string.
1	The application was not connected with a host PS.
4	The PS is busy.
7	An invalid PS position was specified.
9	A system error occurred.
10	Set Cursor not supported on UNIX and VAX hosts

Example

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

WD_StartKSIntercept

This function allows an application to filter any keystrokes sent to a session by a terminal operator. After a call to this function, keystrokes are intercepted and saved until the keystroke buffer overflows or call is made to Function 21, "Reset System," or Function 53, "Stop Keystroke Intercept."

Intercepted keystrokes can be

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

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

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

Syntax

```
DWORD WD_StartKSIntercept(HWND hwnd, PSKSISTRUCT pSKS);
```

Prerequisites

None.

Applicable session parameters

None.

Call parameters

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

Function	WD_StartKSIntercept
hwnd	Window handle of the application
pSKS	Address of a SKSISTRUCT

SKSISTRUCT **format**

Byte	Description
1	Session short name. If blank or null, the session to which the application is currently connected.
2	One of the following characters: D for AID keystrokes only L for all keystrokes
3-6	Reserved.

Return parameters

Result code

Function returns one of the following codes:

Code	Description
0	The function was successful.
1	An invalid PS was specified.
2	An invalid option was specified.
4	The PS is busy.
9	A system error occurred.

Example

```
/* Start intercept of AID keystrokes in session B */
SKISSTRUCT sk;
sk.Shortname = 'B';
sk.Option = 'D';
DWORD Result = WD_StartKSIntercept(hWnd, &sk);
```

WD_GetKey

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

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

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

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

Syntax

```
DWORD WD_GetKey(HWND hwnd, PGKSTRUCT pGKS);
```

Prerequisites

WD_StartKSIntercept

Applicable session parameters

The following session parameters from Function 9 affect this function.

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

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

NWAIT
The function checks for intercepted keystrokes and returns immediately.

Call parameters

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

Function	WD_GetKey
hwnd	Window handle of the application
pGKS	An 8-character code specifying the intercept desired. (See format below.)

Data string format

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

Return parameters

Intercept string

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

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

Typical intercept strings

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

Intercept string	Keyboard equivalent
EAt	"E" represents the session and "A" informs your WD_API application that the keystrokes will be received as ASCII; the returning key is a lowercase "t" (Bytes 6-7 = X'00').
EM@2	"E" represents the session, "M" indicates that the keystrokes will be returned as key mnemonics, and "@2" indicates the key being returned is PF2 (Bytes 8–9 = X'00').
EM@A@Q	"E" represents the session, "M" indicates the keystrokes will be returned as mnemonics, and "@A@Q" indicates that the key being received is ATTN. (Bytes 10–11 = X'00').

Result code

Function returns one of the following codes:

Code	Description
0	The function was successful.
1	An invalid PS was specified.
8	Function 50, "Start Keystroke Intercept," was not called prior to calling this function for the specified PS.
9	A system error occurred.
20	An undefined keystroke combination was entered.
25	The requested keystrokes are not available.
31	The keystroke queue overflowed and keystrokes were lost.

Example

```
/* Allocate space for returned key */
GKSTRUCT gk;
/* Short name of session */
gk.Shortname = 'B';
DWORD Result = WD_GetKey(hWnd, &gk);
```

WD_PostInterceptStatus

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

Syntax

```
DWORD WD_PostInterceptStatus(HWND hwnd, PPISSTRUCT pPPI);
```

Prerequisites

WD_StartKSIntercept

Applicable session parameters

None.

Call parameters

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

Function	WD_PostInterceptStatus
hwnd	Window handle of the application
pPPI	A 2 byte string specifying status to be posted. (See format below.)

Data string format

Byte	Description
1	Session short name. If blank or null, the session to which the application is currently connected.
2	One of the following: • A for accepted keystrokes • R for rejected keystrokes

Return parameters

Result code

Function returns one of the following codes:

Code	Description
0	The function was successful.
1	An invalid PS was specified.
2	An invalid option was specified.
8	Function 50, “Start Keystroke Intercept,” was not called prior to calling this function for the specified PS.
9	A system error occurred.

Example

```
DWORD Result = WD_PostInterceptStatus(hWnd, "BR");
```

WD_StopKSIntercept

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

Syntax

```
DWORD WD_StopHostNotification(HWND hwnd, LPSTR szShortname);
```

Prerequisites

WD_StartKSIntercept

Applicable session parameters

None.

Call parameters

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

Function	WD_StopKSIntercept
hwnd	Window handle of the application
szShortname	A 1-byte string of which the first character is the session short name or a blank/null character indicating a request for the currently connected PS.

Return parameters

Result code

Function returns one of the following codes:

Code	Description
0	The function was successful.
1	An invalid session short name was specified.
8	Function 50, "Start Keystroke Intercept," was not called prior to calling this function for the specified PS.
9	A system error occurred.

Example

```
DWORD Result = WD_StopKSIntercept(hWnd, "B");
```

WD_SendFile

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

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

The program requesting synchronous file transfers must not be intercepting keystrokes for any sessions, must not be awaiting the outcome of another synchronous file transfer, and must not be waiting for host events in any session. Cannnot be used with 5250 sessions.

Syntax

```
DWORD WD_SendFile(HWND hwnd, LPSTR szData);
```

Prerequisites

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

Applicable session parameters

The following session parameters from Function 9 affect this function.

NOQUIET (default) SEND messages are displayed QUIET SEND messages are not displayed.

Call parameters

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

Function	WD_SendFile
hwnd	Window handle of the application
szData	A string (maximum 255 bytes) containing the send command string.

Return parameters

Result code

Function returns one of the following codes:

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

Example

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

WD_ReceiveFile

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

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

The program requesting synchronous file transfers must not be intercepting keystrokes for any sessions, must not be awaiting the outcome of another synchronous file transfer, and must not be waiting for host events in any session. Cannot be used with 5250 sessions.

Syntax

```
DWORD WD_ReceiveFile(HWND hwnd, LPSTR szData);
```

Prerequisites

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

Applicable session parameters

The following session parameters from Function 9 affect this function.

NOQUIET (default) SEND messages are displayed QUIET SEND messages are not displayed.

Call parameters

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

Function	WD_ReceiveFile
hwnd	Window handle of the application
szData	A string (maximum 255 bytes) containing the receive command string. Last byte of the string is EOT if session parameter STREOT is set.

Return parameters

Result code

Function returns one of the following codes:

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

Example

```
/* Receive command string   Assumes     */
/*    PC filename = pcfile.ext          */
/*    Session short name = B            */
/*    Host filename = hostfile.ext      */
/*    CMS transfer options = ASCII,CRLF */
char HllDataStr [] = "pcfile.ext b: hostfile ext (ASCII CRLF";
int Result = WD_ReceiveFile(this.hWnd, HllDataStr);
```

WD_Convert

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

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

Syntax

```
DWORD WD_Convert(HWND hwnd, LPSTR szType, LPPOINT pRC, LPSTR
szPSID);
```

Prerequisites

None.

Applicable session parameters

None.

Call parameters

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

Function	WD_Convert
hwnd	Winow handle of the application
szType	A 2-byte code indicating the type of conversion requested
pRC	Address of a POINT structure.
szPSID	A 1-byte string containing the short name of the session

Convert format

Byte	Description
1	One of the following: • "P" to convert from PS position to row-column coordinates. • "R" to convert from row-column coordinates to PS position.
2	Reserved

RowCol format, convert code P

Integer	Description
1	PS position for which row and column conversion is requested
2	Reserved

RowCol format, convert code R

Integer	Description
1	Row number
2	Column number

Return parameters

Row and column

If converting PS position to row-column coordinates, function replaces the value of call parameter RowCol with a two-integer structure containing the row number in the first integer, and column number in the second. If values returned are zero, the PS position is invalid for the PS.

Result code

Function returns one of the following codes:

Code	Description
0	An invalid PS position or column was specified.
>0	The PS position or column number, depending on the type of conversion being performed.
9998	An invalid session short name was specified.
9999	Second character in data string was not an uppercase “P” or “R.”

Example

```
/* Convert position 199 to row column */
WORD PsPos = 199;
DWORD Result = WD_Convert(hWnd, "P", &PsPos, "B");
```

WD_ConnectWindowServices

This function allows a WD_API application to connect to and manage the PS window.

A WD_API application can be connected to more than one PS window at the same time. The application can switch between windows without having to disconnect.

Only one WD_API application can be connected to a PS window at any one time. Another application can access the PS window only if the first application exits the connection or switches to another PS window connection.

Function 21, "Reset System," reinitializes the WD_API application to its starting point.

Prerequisites

None.

Applicable session parameters

The following session parameters from Function 9 affect this function.

WRITE_SUPER (default)
This parameter is set by a client application program that requires write access and allows only supervisory applications to connect to its PS.

WRITE_WRITE
This parameter is set by a client application program that requires write access and allows other applications that have predictable behavior to connect to its PS.

WRITE_READ
This parameter is set by a client application program that requires write access and allows other applications to use read-only functions on its PS.

WRITE_NONE
This parameter is set by a client application program that requires exclusive access to its PS. No other applications will have access to its PS.

SUPER_WRITE
This parameter is set by a supervisory client application program that allows applications with write access to share the connected PS. The client application program setting this parameter will not cause errors for other applications, but will provide only supervisory-type functions.

WRITE_READ
This parameter is set by a client application program that requires read-only access and allows other applications that perform read-only functions to connect to its PS.

Call parameters

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

Function	WD_ConnectWindowServices
hInstance	Instance handle of the application
Data string	A 1-byte text string containing the session short name.

Return parameters

Result code

Function returns one of the following codes:

Code	Description
0	The function was successful.
1	An invalid session short name was specified.
9	A system error occurred.
10	The function is not supported.
11	The PS was busy.

Example

```
int Result = WD_ConnectWindowServices(this.hWnd, "B");
```

WD_DisconnectWindowServices

This function disconnects the window services connection between an WD_API application and the PS.

Prerequisites

WD_ConnectWindowServices

Applicable session parameters

None.

Call parameters

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

Function	WD_DisconnectWindowServices
hInstance	Instance handle of the application
Data string	A 1-byte text string containing the session short name.

Return parameters

Result code

Function returns one of the following codes:

Code	Description
0	The function was successful.
1	An invalid session short name was specified.
9	A system error occurred.

Example

```
int Result = WD_DisconnectWindowServices(this.hWnd, "B");
```

WD_QueryWindowCoordinates

This function requests the window coordinates of a PS. Window coordinates are returned in pixels.

Prerequisites

WD_ConnectWindowServices

Applicable session parameters

None.

Call parameters

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

Function	WD_QueryWindowCoordinates
hInstance	Instance handle of the application
Data string	A 17-byte data string. (See format below.)

Data string format

Byte	Description
1	Session short name, or blank/null indicating a request for the currently connected PS.
2-17	Reserved

Return parameters

Window coordinates

Function replaces content of call parameter Data string with information about the session window coordinates.

Byte	Description
1	Session short name, or blank/null indicating a request for the currently connected PS.
2–17	Four 32-bit unsigned integers (Xleft, Ybottom, Xright, Ytop) that return the coordinates (in pixels) of a rectangular window relative to the desktop window.

Result code

Function returns one of the following codes:

Code	Description
0	The function was successful.
1	An invalid session short name was specified.
9	A system error occurred.
12	The host session was stopped.

Example

```
char HllDataStr[17];
/* Short name of session */
HllDataStr[0] = 'B';
int Result = WD_QueryWindowCoordinates(this.hWnd, HllDataStr);
```

WD_WindowStatus

This function allows the application to query or change the PS window. The application can change the size, location, or visible state of a PS window. The function can return information regarding the size, location, relative placement, and visible state of a PS window.

Prerequisites

WD_ConnectWindowServices

Applicable session parameters

None.

Call parameters

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

Function	WD_WindowStatus
hInstance	Instance handle of the application
Data string	A 24 or 28-byte data string. (See formats below.)
Data length	24 if extended status is requested; 28 otherwise.
PS position	Not applicable.

Data string format, Set status request

Byte	Description
1	A 1-character session short name.
2	X01 – Set status
3-4	The status set bits. The following codes are valid: • X'0001' — Change window size • X'0002' — Move window • X'0004' — ZORDER window replacement • X'0008' — Set window to visible • X'0010' — Set window to invisible • X'0080' — Activate window • X'0100' — Deactivate window • X'0400' — Minimize window • X'0800' — Maximize window • X'1000' — Restore window
5-8	The X-window position coordinate in pixels. (These bytes are ignored if the move option is not set).
9-12	The Y-window position coordinate in pixels. (These bytes are ignored if the move option is not set).
13-16	The X-window size in pixels. (These bytes are ignored if the size option is not set).
17-20	The Y-window size in pixels. (These bytes are ignored if the size option is not set).
21-24	The window handle for relative window placement. (These bytes are ignored if the ZORDER option is not set.) • X'00000003' — Place window in front of siblings • X'00000004' — Place window behind siblings

Data string format, Query status request

Byte	Description
1	A 1-character session short name.
2	X02 – Query status.
3-4	X'0000'
5-24	Reserved.

Return parameters

Query status result

If the request option (byte 2 of call parameter Data string) was 2 (query status), content of bytes 3 – 16 of call parameter Data string is updated as follows:

Byte	Description
3-4	A word containing a logical OR of bits indicating window state: • X'0008' — The window is visible • X'0010' — The window is invisible • X'0080' — The window is activated • X'0100' — The window is deactivated • X'0400' — The window is minimized • X'0800' — The window is maximized
5-8	The X-window position coordinate.
9-12	The Y-window position coordinate.
13-16	The X-window size in device units.
17-20	The Y-window size in device units.
21-24	The window handle of the session.

Result code

Function returns one of the following codes:

Code	Description
0	The function was successful.
1	An invalid session short name was specified.
2	A parameter error was detected.
9	A system error occurred.
12	The host session was stopped.

Example

```
char HllDataStr[24];
/* Query status for session B */
HllDataStr[0] = 'B';
HllDataStr[1] = 2;
int Result = WD_WindowStatus(this.hWnd, HllDataStr);
```

WD_ChangeWindowName

This function allows the application to change or reset a PS window name.

The exit list processing will reset the name if the application does not do so before exiting. To retain the changed PS name, use Function 102, “Disconnect Window Services.”

Prerequisites

WD_ConnectWindowServices

Applicable session parameters

None.

Call parameters

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

Function	WD_ChangeWindowName
hInstance	Instance handle of the application
Session ID	A 1-byte string containing the session short name
Request	One of the following: • X’01’ — Change the PS window name • X’02’ — Reset the PS window name
PS name	AN ASCII string of 2 to 61 characters including the terminating null

Return parameters

Result code

Function returns one of the following codes:

Code	Description
0	The function was successful.
1	An invalid session short name was specified.
2	A parameter error was detected.
9	A system error occurred.
12	The host session was stopped.

Example

```
/* Change session B PS window name */
int Result = WD_ChangeWindowName(this.hWnd, "B", 1, "Monitor");
```

WD_RunProfile

This function starts a session using its session document (*.EDC file).

Prerequisites

None.

Applicable session parameters

None.

Call parameters

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

Function	WD_QueryWindowCoordinates
hInstance	Instance handle of the application
Profile name	A data string specifying the name and location of the session document.
ShowWindow	Code specifying how the open session is to be displayed (see below).

ShowWindow codes

Code	Description
0	Hides the window and passes activation to another window.
6	Minimizes the specified window and activates the top-level window in the system's list.
9	Activates and displays a window. If the window is minimized or maximized, Windows restores it to its original size and position (Same as 1).
5	Activates a window and displays it in its current size and position.
3	Activates the window and displays it as a maximized window.
2	Activates the window and displays it as an icon.
7	Displays the window as an icon. The current active window remains active.
8	Displays the window in its current state. The current active window remains active.
4	Displays a window in its most recent size and position. The current active window remains active.
1	Activates and displays a window. If the window is minimized or maximized, Windows restores it to its original size and position (Same as 9).

Return parameters

Result code

Function returns one of the following codes:

Code	Description
0	The function was successful.
2	Parameter error, or session document was not found.
9	A system error occurred.

Example

```
/* Open specified EXTRA! session in normal window */
int Result = WD_RunProfile(this.hWnd, "c:\Sessions\HostB.EDC", 1);
```

WD_ShowSession

This function changes the visibility and size of the currently-connected presentation space window.

Prerequisites

WD_ConnectPS

Applicable session parameters

None.

Call parameters

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

Function	WD_SetCursor
hInstance	Instance handle of the application
ShowWindow	Code specifying how the open session is to be displayed (see below).

ShowWindow codes

Code	Description
0	Hides the window and passes activation to another window.
6	Minimizes the specified window and activates the top-level window in the system's list.
9	Activates and displays a window. If the window is minimized or maximized, Windows restores it to its original size and position (Same as 1).
5	Activates a window and displays it in its current size and position.
3	Activates the window and displays it as a maximized window.
2	Activates the window and displays it as an icon.
7	Displays the window as an icon. The current active window remains active.
8	Displays the window in its current state. The current active window remains active.
4	Displays a window in its most recent size and position. The current active window remains active.
1	Activates and displays a window. If the window is minimized or maximized, Windows restores it to its original size and position (Same as 9).

Return parameters

Result code

Function returns one of the following codes:

Code	Description
0	Success; text from the field has been copied to data string.
1	The application was not connected with a host PS.
2	Parameter error

Example

```
/* Activate and maximize the currently-connected PS window */
int Result = WD_SetCursor(this.hWnd, 3);
```

Appendix A: General troubleshooting procedures

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

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

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

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

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

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

Appendix B: Host keyboard mnemonics

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

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

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

Since the Escape character defaults to the at sign (@), you must code the character twice in order to send the escape character as a keystroke. For example, to send a single “@”, you must code “@@”. When your program calls Function 51, “Get Key,” you send a pointer to a keystroke structure used for the returning keystroke. Each keystroke is represented by the following key codes:

- Each key has a number between 1 and 133, which represents the key position on the keyboard.

- Every key has four states: Lower Case, Upper Case, Alt State, and Ctrl State.

Symbols used throughout the tables have the following meanings:

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

Table B-1. Windows keyboard mnemonics

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

Appendix C: Interpreting the Returned Data String for Function 13

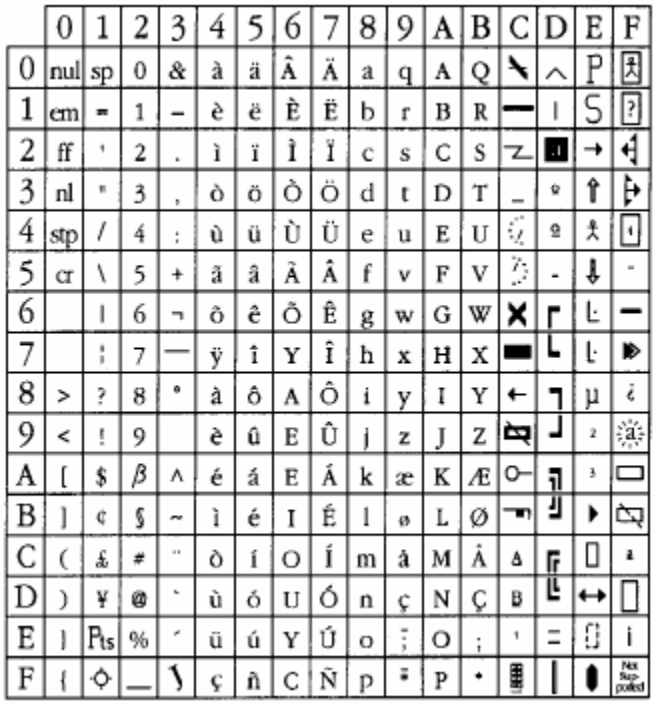
This appendix explains how to decode the data string that Function 13, “Copy OIA,” returns. To interpret this information, you must be able to decipher the OIA image symbols that are returned in positions 2 to 81 of the string, as well as the bits that are returned in positions 82 to 103 of the string.

Position 1 (OIA format byte)

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

Positions 2 to 81 (OIA image symbols)

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



	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	nul	sp	0	&	à	ä	Â	Ä	a	q	A	Q				
1	em	-	1	–	è	ë	È	Ë	b	r	B	R				
2	ff	'	2	.	ì	ï	Ì	Ï	c	s	C	S				
3	nl	"	3	,	ò	ö	Ò	Ö	d	t	D	T				
4	stp	/	4	:	ù	ü	Ù	Ü	e	u	E	U				
5	cr	\	5	+	ã	â	Ã	Â	f	v	F	V				
6			6	¬	õ	ê	Õ	Ê	g	w	G	W				
7			7	—	ÿ	î	Y	Î	h	x	H	X				
8	>	?	8	°	à	ô	A	Ô	i	y	I	Y				
9	<	!	9		è	û	E	Û	j	z	J	Z				
A	[	$	β	^	é	á	E	Á	k	æ	K	Æ				
B	]	¢	§	~	ì	é	I	É	l	ø	L	Ø				
C	(	&	#	¨	ò	í	O	Í	m	å	M	Å				
D	)	¥	@	`	ù	ó	U	Ó	n	ç	N	Ç				
E	]	Pts	%	´	ü	ú	Y	Ú	o	;	O	;				
F			_		ç	ñ	C	Ñ	p		P	•				Not Supported

Positions 82 to 103 (OIA bit groups)

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

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

Group1: Online and screen ownership

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

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

Group 2: Character selection

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

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

Group 3: Shift state

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

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

Group 4: Program symbol support, part 1

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

Bit	Description
0–7	Reserved

Group 5: Highlight, part 1

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

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

Group 6: Color, part 1

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

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

Group 7: Insert

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

Bit	Description
0	Insert mode
1–7	Reserved

Group 8: Input inhibited

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

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

Group 9: Program symbol support, part 2

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

Bit	Description
0–7	Reserved

Group 10: Highlight, part 2

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

Bit	Description
0–7	Reserved

Group 11: Color, part 2

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

Bit	Description
0–7	Reserved

Group 12: Communications error reminder

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

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

Group 13: Printer status error reminder

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

Bit	Description
0–7	Reserved

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

This is the 99th byte in the OIA data.

Bit	Description
0–7	Reserved

Group 15: Reserved

This is the 100th byte in the OIA data.

Bit	Description
0–7	Reserved

Group 16: Automatic key play/record state

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

Bit	Description
0–7	Reserved

Group 17: Automatic key quit/stop state

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

Bit	Description
0–7	Reserved

Group 18: Expanded state

This is the 103rd byte in the OIA data.

Bit	Description
0–7	Reserved

Appendix D: Extended Attributes

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

3270 Character Attributes

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

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

5250 Character Attributes

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

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

Color Attributes

When a subject function is executed with session parameters EAB and XLATE in effect, EAB data are translated in the following format to or from application store:

Bit	Meaning
0–3	Background character color 0000 = Black 0001 = Blue 0010 = Green 0011 = Cyan 0100 = Red 0101 = Magenta 0110 = Brown (3270), Yellow (5250) 0111 = White
4-7	Foreground character color 0000 = Black 0001 = Blue 0010 = Green 0011 = Cyan 0100 = Red 0101 = Magenta 0110 = Brown (3270), Yellow (5250) 0111 = White 1000 = Gray 1001 = Light blue 1010 = Light green 1011 = Light cyan 1100 = Light red 1101 = Light magenta 1110 = Yellow 1111 = White (high intensity)