REXX EHLLAPI *for* EXTRA!

Programmer’s Tool Kit

Part #305713 First Edition: January 1997

The information in this document is subject to change without notice and should not be construed as a commitment by Attachmate Corporation. Attachmate Corporation assumes no responsibility for any errors that may appear in this document. The information disclosed herein is proprietary to Attachmate Corporation and as such, no part of this publication may be reproduced, disclosed, stored in a retrieval system or transmitted in any form or by any means, including electronic, mechanical, photographic or magnetic, without the prior written consent of Attachmate Corporation.

© 1997 Attachmate Corporation. All Rights Reserved. Printed in the USA.

Attachmate and EXTRA! are registered trademarks of Attachmate Corporation.

UNIX is a registered trademark of AT&T Infomation Systems.

VM and CMS are trademarks of and IBM, PS/2, and OS/2 are registered trademarks of International Business Machines Corporation.

Windows is a trademark of Microsoft Corporation.

For help with this product, contact Attachmate's Customer Support Department (See back cover for number).

This manual was written by Heather Killgore.

Table of Contents

Introduction

This guide is written for application developers who want to write REXX programs or procedures that access *EXTRA!* using Extended High Level Language Application Programming Interface (EHLLAPI) services. The *EXTRA!* EHLLAPI interface allows programs to interact with the host 3270 sessions in a way that appears to the host session as a real 3270 terminal user. In other words, the program can read from and write to the host through the terminal session, as well as perform other useful functions.

What Is REXX?

REXX (Restructured Extended Executor) was developed in 1979 as a scripting language for the IBM VM/CMS operating system by Mike Cowlishaw. Since then, REXX has been implemented on many operating systems, including OS/2, UNIX, DOS and Windows 95/NT. REXX provides an alternative to batch files by providing a complete, yet easy-to-use programming language. Because REXX is a system-wide scripting language, it can be used to create multiple REXX-aware programs that interact in a controlled fashion.

What Is EHLLAPI?

EHLLAPI is an interface to *EXTRA!* filled with tools for writing PC-to-mainframe applications. EHLLAPI applications process information between the host and PC, and simulate the actions of a 3270 terminal user.

EHLLAPI consists of a set of defined functions which your application calls. You use these functions to simulate the actions of a person working at a terminal—for example, entering a particular key sequence or choosing a command.

The REXX interface to EHLLAPI under Windows 95 and Windows NT is called SAAHLLAPI.

Why Use REXX EHLLAPI?

REXX applications can reduce operations costs, decrease support time, and simplify training, because nearly any operation performed by a 3270 user can be automated and simplified by using a REXX application using SAAHLLAPI functions.

You develop your REXX SAAHLLAPI programs on your own PC, rather than on a mainframe, saving time and mainframe resources. Your REXX programs are generally shorter and easier to read, debug, and maintain than mainframe programs. In addition, you'll be able to more easily create structured applications in a high level language.

Using REXX and SAAHLLAPI, you can easily write applications that accomplish the following:

- Automate complicated logon procedures, eliminating the need to teach users how to log on or operate mainframe systems.
- Develop custom menus to "front end" for a mainframe application.
- Simplify user screens for complex mainframe applications, shortening the learning curve for new or occasional users.
- Create executive workstations that are less intimidating and provide quicker access to critical data.
- Decrease the number of support calls by providing more help screens for mainframe applications.
- Process data during off-hours, unattended.
- Download master files to a PC to reduce the load on the mainframe (distributed processing).
- Combine multiple mainframe or Windows applications to create a master application that is tailor-made for your business.

Related Publications

Although this guide should allow a person who is moderately proficient with the REXX language to write programs that use EHLLAPI, we do not discuss REXX or *EXTRA!* EHLLAPI in detail. To write more complex REXX EHLLAPI programs, you may need a broader set of documentation, such as one of the following documents:

- IBM *OS/2 Procedures Language REXX Reference* (S01F-0271)
- IBM *OS/2 Procedures Language REXX User's Guide* (S01F-0272)
- Attachmate *EXTRA! EHLLAPI Programmer's Tool Kit*

Installing REXX EHLLAPI Support

The *EXTRA!* Setup program installs the REXX EHLLAPI Dynamic Link Library (SAAHLAPI.DLL) when you install *EXTRA!*. It is installed in the same directory in which you installed *EXTRA!* (the default directory offered during installation is `\Program Files\EPC`).

A REXX application can gain access to EHLLAPI services as external functions. The EHLLAPI services are then loaded into memory only when used. To register the REXX EHLLAPI functions with the REXX interpreter, insert the following line into all REXX procedures using the REXX EHLLAPI interface:

```
if rxfuncquery('hllapi') then
   call rxfuncadd 'hllapi','saahlapi','hllapisrv'
```

The REXX EHLLAPI interface provides a single function name to be used with a set of parameters. The function name is the first parameter of the RXFUNCADD call above, and defaults to HLLAPI.

Overview of EHLLAPI Function Calls and Return Values

Calling REXX EHLLAPI Functions

Your REXX application can call REXX EHLLAPI as either a REXX function or a REXX subroutine.

Using a REXX Function Call

```
HLLAPI( function-name, parameters )
```

The function which returns a value which must be assigned to a variable, and is assigned by default to the REXX variable *rc*. For example:

```
rc = HLLAPI('wait')
```

Using a REXX Subroutine Call

```
call HLLAPI function-name, parameters
```

The subroutine places the return value in the REXX RESULT variable.

Function Reference Conventions

All of the functions in this chapter are presented in the same manner. The available SAAHLLAPI functions are listed in this guide in alphabetical order. First, we list the function name, followed by a brief description of the function's purpose and the following information.

Prerequisite

Most functions require another function that must be called and successfully completed before the desired call is issued. If *None* appears, no prerequisite calls are necessary.

Syntax

This area presents the call as it should be written in an application. The syntax definition uses variables to represent the calling parameters.

Call Parameters

This area lists the parameters that must be presented in a call statement before your program can call a REXX SAAHLLAPI function. The call parameters must be declared as specified in the function definition.

Return Codes

This area lists the information that your program will receive from the resident module after your function has been processed. A function can return a numeric code or a data string. The returned values are discussed in the explanation of the individual functions.

Programming Notes

In the REXX environment, the *EXTRA!* EHLLAPI subsystem does not automatically disconnect and reset a connected session when the REXX procedure ends. This means that if the REXX application ends without disconnecting a session, the session will remain connected and any options set by Set_Session_Parms will remain in effect.

It is your responsibility as a REXX programmer to make sure that the REXX program properly disconnects and resets sessions under all program termination conditions, including error conditions.

The REXX EHLLAPI interface does not support all of the functions available in the *EXTRA!* EHLLAPI interface. These limitations are primarily due to the inability of REXX to dynamically allocate shared memory and to process semaphores. The EHLLAPI functions not supported in the REXX EHLLAPI subset are:

- Structured Fields
- Asynchronously executing function calls
- EHLLAPI Storage Manager
- Execution search path for SEND.EXE and RECEIVE.EXE
- Lock_PMSVC and Lock_PS

REXX EHLLAPI Function Reference

Change_Switch_Name

This function changes or resets the name of the session listed on the host window title bar.

Prerequisite

Connect_PM.

Syntax

```
HLLAPI( ’Change_Switch_Name’, session_id, type [, new_name ] )
```

Call Parameters

Parameter	Description
session_id	The single-character short name of the session be renamed on the title bar. If necessary, use the **Query_Sessions** function to determine the short names of the sessions that are configured on the system.
type	One of the following: • **Set** renames the session name (title) from session_id to new_name. • **Reset** restores the original session name (title). Note: Only the first character of **Set** or **Reset** is significant.

Return Codes

The Change_Switch_Name function returns one of the following to the REXX variable rc:

Code	Description
0	The Change_Switch_Name function was successful.
1	Your program is not currently connected to the host.
2	An error was made in specifying parameters.
9	A system error occured.
12	The session stopped.

Change_Window_Name

This function changes or resets the name of the session listed on the host window Title Bar.

Prerequisite

Connect_PM.

Syntax

```
HLLAPI( ’Change_Window_Name’, session_id, type [, new_name ] )
```

Call Parameters

Parameter	Description
session_id	The single-character short name of the session be renamed on the Title Bar. If necessary, use the **Query_Sessions** function to determine the short names of the sessions that are configured on the system.
Type	One of the following: • **Set** renames the window name (title) from session_id to new_name. • **Reset** restores the original window name (title). Note: Only the first character of **Set** or **Reset** is significant.

Return Codes

The Change_Window_Name function returns one of the following to the REXX variable rc:

Code	Description
0	The Change_Window_Name function was successful.
1	Your program is not currently connected to the host session.
2	An error was made in specifying parameters.
9	A system error occured.
12	The session stopped.

Connect

This function connects the REXX procedure to the presentation space (PS).

Prerequisite

None.

Syntax

```
HLLAPI( 'Connect', session_id )
```

Call Parameters

Parameter	Description
session_id	The single-character short name of the session to which you want to connect. If necessary, use the Query_Sessions call to determine the short names of the sessions that are configured on the system, or check the configuration in the Executive.

Return Codes

The Connect function returns one of the following to the REXX variable rc:

Code	Description
0	OK, connected
1	Invalid session_id
4	Connect, session busy
5	Connect, session locked
9	System error
11	Session is already connected to another application

Connect_PM

This function connects the REXX procedure to the presentation space window (configured HLLAPI session window).

Prerequisite

None.

Syntax

```
HLLAPI( 'Connect_PM', session_id )
```

Call Parameters

Parameter	Description
session_id	The single-character short name of the session window that you want to connect to.

Return Codes

The Connect_PM function returns one of the following to the REXX variable rc:

Code	Description
0	The Connect_PM function was successful.
1	Your program is not currently connected to the host session.
9	A system error occurred.
10	The function is not supported by your emulation program.
11	The session is already being used by another system function.

Convert_Pos

Converts column/row to PS position, or PS position to column/row for the session *session_id*. The valid values for column/row are the size specified when this session was configured. For example, a session with 24 rows and 80 columns contains positions 1 to 1920.

Note: Column is the first parameter specified when column/row conversion is being requested, and column is the first value returned when position conversion is requested.

Prerequisite

None.

Syntax

```
HLLAPI( 'Convert_pos', session_id , column | position , row )
```

Call Parameters

Parameter	Description
session_id	The single-character short name of the session. If necessary, use the Query_Sessions call to determine the short names of the sessions that are configured on the system, or check the configuration in the Executive.
Column | position	Either the number of the column for which you are requesting a conversion to position number, or the number of the PS position you want converted to a column/row value.
Row	The number of the row for which you are requesting a conversion to position number. Omit this parameter when converting from position number to column/row.

Return Codes

Column/row conversion

When converting column/row to PSP, the Convert_Pos function returns one of the following to the REXX variable *rc*:

Code	Description
0	Column or row specified is outside PS
n	The position number of the specified column and row

For example:

```
HLLAPI( Convert_pos, 'a', 10 , 2 ) == '90'
```

converts column 10, row 2, to PSP 90 in a 24x80 PS (session A).

Position conversion

When converting PSP to column/row, the Convert_Pos function returns one of the following to the REXX variable *rc*:

Code	Description
0	Position specified is outside the PS
c r	*c* is the column number and *r* is the row number for the specified position

For example:

```
HLLAPI( Convert_pos, 'a', 90 ) == '10 2'
```

converts position 90 to column 10, row 2 in a 24x80 PS (session A).

Copy_Field_To_Str

This function returns the data located at field position *pos* in the connected PS, for length *length*. The *pos* and *length* values can be found by using the Find_Field_Pos and Find_Field_Len commands.

Prerequisite

Connect

Syntax

```
HLLAPI( ’Copy_field_to_str’ , pos , length )
```

Call Parameters

Parameter	Description
pos	The position of the field that you want to copy
length	The length (in bytes) of the field that you want to copy

Return Codes

The Copy_Field_To_Str function returns one of the following to the REXX variable *rc*:

Code	Description
'' (Null string)	No field data at *pos*, or invalid *pos*
Field Data	The data of length *length* or up to the end of the field at *pos*

Copy_OIA

This function returns the image of the Operator Information Area from the connected session, along with detail information in the OIA group area.

Prerequisite

Connect

Syntax

```
HLLAPI( ’Copy_OIA’ )
```

Call Parameters

None.

Return Codes

The Copy_OIA function returns one of the following to the REXX variable *rc*:

Code	Description	
'' (Null string)	Not connected or other error	
OIA Data	A 103-byte copy of the OIA data consisting of:	
	Position	**Description**
	1	OIA format byte
	2–81	The OIA image in hex
	82–103	A 22-byte OIA group field

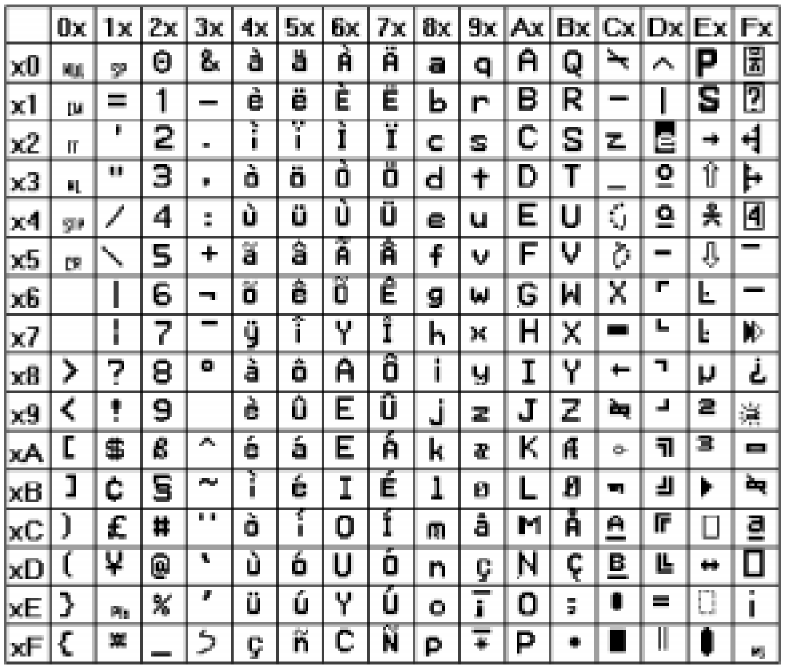
Interpreting Returned Data

This section explains how to decode the data string that the Copy_OIA function returns. To interpret this information, you must be able to decipher the OIA image symbols that are returned in positions 2–81 of the string, as well as the bits that are returned in positions 82–103 of the string. The symbols that are returned in Positions 2–81 of its data string can be interpreted by looking up the desired hexadecimal value below in the table in the section "The OIA Image Symbols in Positions 2–81."

Position 1

Position 1 of the returning data string always returns the format byte. The function copies the X clock from position 5 on the OIA.

The OIA Image Symbols in Positions 2–81



	0x	1x	2x	3x	4x	5x	6x	7x	8x	9x	Ax	Bx	Cx	Dx	Ex	Fx
x0	NUL	SP	Θ	&	à	ä	À	Ä	a	q	A	Q	[illegible]	^	P	[illegible]
x1	LM	=	1	–	è	ë	È	Ë	b	r	B	R	–	|	S	⍰
x2	IT	'	2	.	ì	ï	Ì	Ï	c	s	C	S	z	[illegible]	→	[illegible]
x3	NL	"	3	·	ò	ö	Ò	Ö	d	+	D	T	_	[illegible]	⇧	[illegible]
x4	STP	/	4	:	ù	ü	Ù	Ü	e	u	E	U	[illegible]	[illegible]	[illegible]	[illegible]
x5	CR	\	5	+	ã	â	Ã	Â	f	v	F	V	[illegible]	–	⇩	‾
x6		|	6	¬	õ	ê	Õ	Ê	g	w	G	W	X	┌	[illegible]	—
x7		¦	7	¯	ÿ	î	Y	Î	h	x	H	X	▬	└	[illegible]	[illegible]
x8	>	?	8	°	à	ô	A	Ô	i	y	I	Y	←	┐	µ	¿
x9	<	!	9		è	û	E	Û	j	z	J	Z	[illegible]	┘	²	[illegible]
xA	[	$	ß	^	é	á	E	Á	k	æ	K	Æ	∘	╗	³	▭
xB	]	¢	§	~	ì	é	I	É	l	ø	L	Ø	[illegible]	╝	▶	[illegible]
xC	)	£	#	¨	ò	í	O	Í	m	å	M	Å	A̲	╔	▯	a̲
xD	(	¥	@	`	ù	ó	U	Ó	n	ç	N	Ç	B̲	╚	↔	▯
xE	}	Pts	%	´	ü	ú	Y	Ú	o	ī	O	;	●	=	▯	i
xF	{	※	_	[illegible]	ç	ñ	C	Ñ	p	[illegible]	P	•	█	‖	●	[illegible]

The table above displays the symbols found in the DFT host, and CUT host presentation spaces. These symbols can be part of the OIA image returned in positions 2–81 of the returning data string.

The remainder of the returning data string (positions 82–103) can provide more detail about the symbols returned in positions 2–81. The table below should help you map the returned symbols to the groups covered in the next section.

This image returned in this data string position	Is expanded upon in
Position 2	Group 1 (position 82)
Position 10	Group 8 (positions 89–93)
Position 37	Group 2 (position 83)
Position 43	Group 3 (position 84)
Position 48	Group 5 (position 86)
Position 50	Group 6 (position 87)
Position 53	Group 7 (position 88)

Returning OIA Group Indicators in Positions 82–103

The remaining positions in the returning data string can be interpreted with the help of the following tables. Each position or group returns a bit number that explains a particular OIA characteristic.

Group	Characteristic Explained	Position number
1	On-Line and Screen Ownership	82
2	Character Selection	83
3	Shift State	84
4	PSS Group 1	85
5	Highlight Group 1	86
6	Color Group 1	87
7	Insert	88
8	Input Inhibited (5 bytes)	89–93
9	PSS group 2	94
10	Highlight Group 2	95

Group	Characteristic Explained	Position number
11	Color Group 2	96
12	Communication Error Reminder	97
13	Printer Status	98
14	Reserved Group	99
15	Reserved Group	100
16	Autokey Play/Record Status	101
17	Autokey Abort/Pause State	102
18	Enlarge State Position	103

	Bit 0	Bit 1	Bit 2	Bit 3	Bit 4	Bit 5	Bit 6	Bit 7
Group 1 (Data String position 82)	Setup Mode	Test Mode	SSCP-LU session owns screen	LU-LU session owns screen	Online and not owned	Subsys-tem ready	Rsvd.	Rsvd.
Group 2 (Data String position 83)	Extended select	APL	Kana	Alpha	Text	Rsvd.	Rsvd.	Rsvd.
Group 3 (Data String position 84)	Numeric	Upper shift	Rsvd.	Rsvd.	Rsvd.	Rsvd.	Rsvd.	Rsvd.
Group 4 (Data String position 85)	User-select-able	Field inherit	Rsvd.	Rsvd.	Rsvd.	Rsvd.	Rsvd.	Rsvd.
Group 5 (Data String position 86)	User-select-able	Field inherit	Rsvd.	Rsvd.	Rsvd.	Rsvd.	Rsvd.	Rsvd.
Group 6 (Data String position 87)	User-select-able	Field inherit	Rsvd.	Rsvd.	Rsvd.	Rsvd.	Rsvd.	Rsvd.

Continued on next page.

3270 OIA Group Indicators (cont.)

	Bit 0	Bit 1	Bit 2	Bit 3	Bit 4	Bit 5	Bit 6	Bit 7
Group 7 (Data String position 88)	Insert mode	Rsvd.	Rsvd.	Rsvd.	Rsvd.	Rsvd.	Rsvd.	Rsvd.
Group 8 Byte 1 (Data String position 89)	Non-resetable machine check	Reserve for security key	Machine check	Commu-nications check	Program check	Retry	Device not working	Device very busy
Byte 2 (Data String position 90)	Device busy	Terminal wait	Minus symbol	Minus function	Too much entered	Not enough entered	Wrong number	Numeric field
Byte 3 (Data String position 91)	Rsvd.	User not author-ized	User not author-ized (minus function)	Invalid dead key combi-nation	Wrong place	Rsvd.	Rsvd.	Rsvd.
Byte 4 (Data String position 92)	Message pending	Partition wait	System Wait	Hard-ware mis-match	Logical terminal not config-ured at control unit	Rsvd.	Rsvd.	Rsvd.
Byte 5 (Data String position 93)	Autokey inhibit	Applica-tion program has inhibited operator input	Rsvd.	Rsvd.	Rsvd.	Rsvd.	Rsvd.	Rsvd.
Group 9 (Data String position 94)	PC selected	PC display disabled	Rsvd.	Rsvd.	Rsvd.	Rsvd.	Rsvd.	Rsvd.
Group 10 (Data String position 95)	Selected	Rsvd.	Rsvd.	Rsvd.	Rsvd.	Rsvd.	Rsvd.	Rsvd.
Group 11 (Data String position 96)	Selected	Rsvd.	Rsvd.	Rsvd.	Rsvd.	Rsvd.	Rsvd.	Rsvd.
Group 12 (Data String position 97)	Commu-nications Error	Re-sponse time monitor	Rsvd.	Rsvd.	Rsvd.	Rsvd.	Rsvd.	Rsvd.

Continued on next page.

3270 OIA Group Indicators (cont.)

	Bit 0	Bit 1	Bit 2	Bit 3	Bit 4	Bit 5	Bit 6	Bit 7
Group 13 (Data String position 98)	Print code not custom-ized	Printer malfunc-tion	Printer printing	Assign printer	What printer	Printer assign-ment	Rsvd.	Rsvd.
Group 14 (Data String position 99)	Rsvd.	Rsvd.	Rsvd.	Rsvd.	Rsvd.	Rsvd.	Rsvd.	Rsvd.
Group 15 (Data String position 100)	Rsvd.	Rsvd.	Rsvd.	Rsvd.	Rsvd.	Rsvd.	Rsvd.	Rsvd.
Group 16 (Data String position 101)	Play	Record	Rsvd.	Rsvd.	Rsvd.	Rsvd.	Rsvd.	Rsvd.
Group 17 (Data String position 102)	Record-ing overflow	Pause	Rsvd.	Rsvd.	Rsvd.	Rsvd.	Rsvd.	Rsvd.
Group 18 (Data String position 103)	Window is enlarged	Rsvd.	Rsvd.	Rsvd.	Rsvd.	Rsvd.	Rsvd.	Rsvd.

Copy_PS

This function returns the entire contents of the PS of the currently connected session.

Prerequisite

Connect

Syntax

```
HLLAPI( ’Copy_PS’ )
```

Call Parameters

None.

Return Codes

The Copy_PS function returns one of the following to the REXX variable *rc*:

Code	Description
'' (Null string)	Not connected or other error
PS Data	The content of the returned data is dependent on the setting of the EAB (Extended Attribute Bytes) value in the Set_Session_Parms option: If EAB is OFF, only the text of the PS will be returned. If EAB is ON, 2 bytes are returned for each displayable byte on the screen. The first byte contains the EAB value, the second byte contains the text data.

Copy_PS_To_Str

This function returns data from the currently connected session from position *pos* for length *length*.

Prerequisite

Connect

Syntax

```
HLLAPI( 'Copy_PS_to_str' , pos, length )
```

Call Parameters

Parameter	Description
pos	The beginning position of the data you want to copy
length	The number of characters in the string you are copying

Return Codes

The Copy_PS_To_Str function returns one of the following to the REXX variable *rc*:

Code	Description
'' (Null string)	Not connected or other error
PS data string	The content of the returned data is dependent on the setting of the EAB (Extended Attribute Bytes) value in the Set_Session_Parms option: If EAB is OFF, only the text of the PS will be returned. If EAB is ON, 2 bytes are returned for each displayable byte on the screen. The first byte contains the EAB value, the second byte contains the text data.

Copy_Str_To_Field

This function copies *string* to the field at position *pos* in the currently connected session.

Prerequisite

Connect

Syntax

```
HLLAPI( 'Copy_str_to_field', string , pos )
```

Call Parameters

Parameter	Description
string	The string you are copying to the session field
pos	The position of the field to which you want to copy

Return Codes

The Copy_Str_To_Field function returns one of the following to the REXX variable *rc*:

Code	Description
0	OK, string copied
1	Not connected
2	Parameter error
5	Protected field or bad string data (such as field attribute)
6	Data copied, but truncated because string was longer than the field
7	Parameter *pos* is invalid
24	The screen has no fields (unformatted)

Copy_Str_To_PS

This function copies *string* to position *pos* in the connected session.

Prerequisite

Connect

Syntax

```
HLLAPI( ’Copy_str_to_PS’, string , pos )
```

Call Parameters

Parameter	Description
string	The string you are copying to the session
pos	The position in the connected session to which you want to copy

Return Codes

The Copy_Str_To_PS function returns one of the following to the REXX variable rc:

Code	Description
0	OK, string copied
1	Not connected
2	Parameter error
5	Protected field or bad string data (such as field attribute)
6	Data copied, but truncated because string was longer than the field
7	Parameter *pos* is invalid

Disconnect

This function disconnects from the currently connected session.

Prerequisite

Connect

Syntax

```
HLLAPI( 'Disconnect' )
```

Call Parameters

None.

Return Codes

Code	Description
0	OK, disconnected
1	Can't disconnect because not currently connected

Disconnect_PM

This function disconnects from the session window.

Prerequisite

Connect_PM

Syntax

```
HLLAPI( 'Disconnect_PM', session_id )
```

Call Parameters

Parameter	Description
session_id	The single-character short name of the session you want to connect to.

Return Codes

Code	Description
0	The Disconnect_PM function was successful.
1	Your program is not currently connected to the host presentation space.
9	A system error occured

Find_Field_Len

This function returns the length of the field with attributes *search_option* starting at position *pos* within the current field.

Prerequisite

Connect

Syntax

```
HLLAPI( 'Find_field_len' , search_option , pos )
```

Call Parameters

In the options below, the "^" character represents a space or blank passed as part of the data string. For example, passing a data string of two spaces indicates that you are searching for the length of the current field.

Parameter	Description	
search option	'^^' or 'T^'	Current field (the field that the cursor is in)
	'N^'	Next field, either protected or unprotected
	'P^'	Previous field, either protected or unprotected
	'NP'	Next protected field
	'NU'	Next unprotected field
	'PP'	Previous protected field
	'PU'	Previous unprotected field
pos	The screen position at which you want to begin searching	

Return Codes

The Find_Field_Len function returns one of the following to the REXX variable *rc*:

Code	Description
0	Specified field not found
>0	Length of the specified field

Find_Field_Pos

This function returns the position of the field with attributes search_option starting at position *pos* within the PS.

Prerequisite

Connect

Syntax

```
HLLAPI( 'Find_field_pos' , search_option , pos )
```

Call Parameters

In the options below, the "∧" character represents a space or blank passed as part of the data string. For example, passing a data string of two spaces indicates that you are trying to find the PS position of the beginning of the current field.

Parameter	Description	
search option	'∧∧' or 'T∧'	Current field (the field that the cursor is in)
	'N∧'	Next field, either protected or unprotected
	'P∧'	Previous field, either protected or unprotected
	'NP'	Next protected field
	'NU'	Next unprotected field
	'PP'	Previous protected field
	'PU'	Previous unprotected field
pos	The screen position at which you want to begin searching	

Return Codes

The Find_Field_Pos function returns one of the following to the REXX variable *rc*:

Code	Description
0	Specified field not found
n	Numeric position of the specified field

Get_Key

This function is used to get a keystroke from the specified *session_id*, or from the currently connected session if *session_id* is blank. If no keystroke is waiting in the session, the program waits until a keystroke becomes available.

If keystroke interception is active (using Start_Keystroke_Intercept), no keystrokes are sent to the connected session until two conditions are met.

1. A Get_Key operation is used to remove the keystroke from the intercept buffer.
2. An Intercept_Status operation is used to either 'Accept' or 'Reject' the keystroke. If the status operation is 'Accept,' the keystroke can be sent (using Sendkey) to the connected session, otherwise the keystroke will be thrown away and a beep will sound.

Prerequisite

Start_Keystroke_Intercept

Syntax

```
HLLAPI( 'Get_key' , session_id )
```

Call Parameters

Parameter	Description
session_id	The single-character short name of the session. If necessary, use the Query_Sessions call to determine the short names of the sessions that are configured on the system, or check the configuration in the Executive.

Return Codes

The Get_Key function returns one of the following to the REXX variable *rc*:

Code	Description
'' (Null string)	Error or not connected to *session_id*
Data String	The contents of string depend on the keys pressed by the 3270 session operator, for example: **Length** — **Data** 1 — ASCII code for key, for example, ‘a’ 2 — Keystroke mnemonic, for example, ‘@E’ (Enter) 3 — Keystroke mnemonic, for example, ‘@ra’ (Ctrl+a) 6 — Keystroke mnemonic, for example, ‘@r@A@1’ (Ctrl+Alt+F1)

Note: The ‘@’ escape character is set using the Set_Session_Parms call, ESC=.

Get_Window_Status

This function returns the current window status as a string of ASCII characters in hexadecimal format.

Prerequisite

Connect_PM

Syntax

```
HLLAPI( 'Get_window_status' , session_id )
```

Call Parameters

Parameter	**Description**
session_id	The single-character short name of the session.

Return Codes

The Get_Window_Status function returns one of the following to the REXX variable *rc*:

Code	**Description**
''	Null. Not connected to the PM window. Refer to the Connect_PM function for more information..
0008	The window is visible.
0010	The window is invisible.
0080	The window is activated.
0100	The window is deactivated.
0400	The window is minimized.
0800	The window is maximized.

Intercept_Status

This function is used to signal *session_id* as to the disposition of a keystroke obtained by Get_Key.

Prerequisite

Start Keystroke Intercept

Syntax

```
HLLAPI( 'Intercept_status', session_id, status )
```

Call Parameters

Parameter	Description
session_id	The single-character short name of the session. If necessary, use the Query_Sessions call to determine the short names of the sessions that are configured on the system, or check the configuration in the Executive.
status	'A' Accept the keystroke 'R' Reject the keystroke and beep

Return Codes

The Intercept_Status function returns one of the following to the REXX variable *rc*:

Code	Description
0	OK, keystroke accepted/rejected
1	Not connected
8	No Start_Keystroke_Intercept active
9	System error

Pause

This function causes a timed pause of *n* half-second intervals to occur.

If the Set_Session_Parms call (described later) set IPAUSE and you have done a Start_Host_Notify call, the pause will also be ended by an update to the host screen. If session_name is provided and IPAUSE has been set, only updates to the specified session will interrupt the pause. Otherwise, updates to any connected session will interrupt the pause (if IPAUSE has been set).

Prerequisite

None.

Syntax

```
HLLAPI( ’Pause’ , pause_length , session_name )
```

Call Parameters

Parameter	Description
pause_length	The number of half-seconds that you want the pause to last
session_name	A two-character string. The first character is the short session id (A–Z), and the second character is the # character (number sign or pound sign), coded exactly as shown. For example, to pause session B for five seconds you would code: `HLLAPI( ’Pause’ , 10 , B# )`

Return Codes

The Pause function returns one of the following to the REXX variable *rc*:

Code	Description
0	OK, pause duration completed
9	System error
26	Host session has been updated. Use the Query_Host_Update function to obtain more information.

Query_Close_Intercept

This function determines if a close request was initiated from the session.

Prerequisite

Start_Close_Intercept

Syntax

```
HLLAPI( 'Query_close_intercept' , session_id )
```

Call Parameters

Parameter	Description
session_id	The single-character short name of the host session.

Return Codes

The Query_Close_Intercept function returns one of the following to the REXX variable *rc*:

Code	Description
0	A close intercept event did not occur.
1	Your program is not currently connected to the host session.
2	An error was made in specifying parameters.
8	No prior Start_Close_Intercept function was called for this host presentation space.
9	A system error occurred.
12	The session stopped.
26	A close intercept occurred since the last Query_Close_Intercept function call.

Query_Cursor_Pos

This function returns the cursor position in the currently connected session.

Prerequisite

Connect

Syntax

```
HLLAPI( 'Query_cursor_pos' )
```

Call Parameters

None.

Return Codes

The Query_Cursor_Pos function returns one of the following to the REXX variable *rc*:

Code	Description
0	Session not connected
>0	Cursor position

Query_Field_Attr

This function returns the hexadecimal representation of the attribute of the field located at position *pos* in the currently connected session.

Prerequisite

Connect

Syntax

```
HLLAPI(  Query_field_attr' , pos )
```

Call Parameters

Parameter	Description
pos	The position of the field from which you are returning an attribute

Return Codes

The Query_Field_Attr function returns one of the following to the REXX variable *rc*:

Code	Description
0	Session not connected or unformatted screen
Data string	Attribute bytes (printable hexadecimal characters equal or greater than C0)

Query_Host_Update

This function determines if the OIA or PS for session *session_id* has been updated.

Prerequisite

Start_Host_Notify

Syntax

```
HLLAPI( 'Query_host_update', session_id )
```

Call Parameters

Parameter	Description
session_id	The single-character short name of the session. If necessary, use the Query_Sessions call to determine the short names of the sessions that are configured on the system, or check the configuration in the Executive.

Return Codes

The Query_Host_Update function returns one of the following to the REXX variable *rc*:

Code	Description
0	No update since last call
1	Not connected to a session
8	Host notification has not been requested (see the Start_Host_Notify call later in this command reference)
9	System error
21	The OIA has been updated
22	The PS has been updated
23	Both OIA and PS have been updated

Query_Session_Status

This function returns various status information for session *session_id*, or from the currently connected session if *session_id* is blank.

Prerequisite

None.

Syntax

```
HLLAPI( 'Query_session_status' , session_id )
```

Call Parameters

Parameter	Description
session_id	The single-character short name of the session. If necessary, use the Query_Sessions call to determine the short names of the sessions that are configured on the system, or check the configuration in the Executive.

Return Codes

The Query_Session_Status function returns one of the following to the REXX variable *rc*:

Code	Description
'' (Null string)	Not connected
Data String	18 bytes as follows: 1-byte session short name 8-byte session long name in ASCII 1-byte session type: 'D' 3270 terminal session 'E' 3270 printer session 'P' Personal Computer 1-byte binary number containing Control Program features: **Bit** **Description** 0 EAB (0=basic, 1=EABs) 1 PSS (0=no symbol support, 1=supports programmed symbols) 2–7 Reserved 2-byte binary number (in Intel reversed format) containing the number of rows in the session 2-byte binary number (in Intel reversed format) containing the number of columns in the session 2-byte binary number (in Intel reversed format) containing the host code page value 1 byte reserved

Note: To get the decimal values for the last three fields (row, col, codepage), use c2d(reverse(x)) after parsing them out of the string.

Query_Sessions

This function returns 12 bytes of status information for each configured session or '' (null string) if an error occurred.

Prerequisite

None.

Syntax

```
HLLAPI( 'Query_sessions' )
```

Call Parameters

None.

Return Codes

The Query_Sessions function returns one of the following to the REXX variable *rc*:

Code	Description
'' (Null string)	Not connected
Data String	12 bytes as follows: 1-byte session short name 8-byte session long name in ASCII 1-byte session type - 'H' (host) or 'P' (personal computer) 2-byte binary number (in Intel reversed format) containing the PS size for the session

Note: To get the decimal value for the last field (pssize), use c2d(reverse(x)) after parsing it out of the string.

Query_System

This function returns a 35-byte system configuration string or '' (null string) if an error occurs.

Prerequisite

None.

Syntax

```
HLLAPI( 'Query_system' )
```

Call Parameters

None.

Return Codes

The Query_System function returns one of the following to the REXX variable *rc*:

Code	Description
'' (Null string)	Not connected
Data String	35 bytes as follows: 1-byte EHLLAPI version (in ASCII) 2-byte EHLLAPI level (in ASCII) 6-byte MMDDYY EHLLAPI date (in ASCII) 1-byte hardware base, 'Z' means system bytes are valid. 'U' means system bytes NOT valid. 1 byte of 'X' 2-byte binary (stored in normal Intel reverse order) sequence number 2 bytes (ASCII) of the control program number 1 byte of '1' 4-byte Extended Error code 1 (in ASCII) 4-byte Extended Error code 2 (In ASCII)

Return Codes (cont.)

Code	Description
Data String (cont.)	2-byte binary (in Intel reverse order) hardware base and sub-model types (this is a partial list of possible return values for this field): x'FC01' PC/AT-339 x'FC02' XT/286 x'FC05' PS/2 Model 60 x'F800' PS/2 Model 80 (16 Mhz) x'F801' PS/2 Model 80 (20 Mhz) x'F804' PS/2 Model 70-121 x'F809' PS/2 Model 70-E61,F61 x'F80B' PS/2 Model P70 x'F80C' PS/2 Model 55SX x'F80D' PS/2 Model 70-A21 x'F816' PS/2 Model 90 x'F819' PS/2 Model 35,40 2-byte binary (in Intel reverse order) of the PC code page being used 1-byte value identifying the display type being used (this is a partial list of possible return values for this field): 'C' CGA 'E' EGA 'V' VGA 'H' 8514 'A' 8503 'U' Unknown

Note: To get the decimal value for the last field (pssize), use c2d(reverse(x)) after parsing it out of the string.

Query_Window_Coord

This function requests the window coordinates from the window for the host session, or from the currently connected session if session_id is blank.

Prerequisite

Connect_PM.

Syntax

```
HLLAPI( 'Query_window_coord', session_id )
```

Call Parameters

Parameter	Description
session_id	The single-character short name of the session window..

Return Codes

The Query_Window_Coord function returns one of the following to the REXX variable *rc*:

Return Code	Description
''	Null. Not connected.
Data	The data string returns 4 decimal numbers in the following format: xLeft yBottom xRight yTop

Receive_File

This function executes the RECEIVE.EXE program as if it had been entered at the command line. The contents of string must be an acceptable syntax for the receive command.

Prerequisite

None.

Syntax

```
HLLAPI( 'Receive_file', string )
```

Call Parameters

Parameter	Description
string	The parameters you wish to pass to the RECEIVE.EXE program. The parameters for the RECEIVE command are: `<pc file name> <session id> <host file name> <options>` For more information on file transfer commands and options, see Appendix A.

Return Codes

The Receive_File function returns one of the following to the REXX variable *rc*:

Code	Description
2	Parameter error
3	File transfer complete
4	File transfer complete, records segmented
9	System error
27	Terminated by ^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

Release

This function unlocks the connected display session keyboard for user input.

Prerequisite

Connect

Syntax

```
HLLAPI( ’Release’ )
```

Call Parameters

None.

Return Codes

The Release function returns one of the following to the REXX variable *rc*:

Code	Description
0	OK, released
1	Not connected

Note: If you disconnect while the keyboard is locked (using RESERVE), the keyboard is released automatically.

Reserve

This function locks the keyboard of the currently connected session from user input. The keyboard remains locked until either a Release or a Disconnect is done.

Prerequisite

Connect

Syntax

```
HLLAPI( 'Reserve' )
```

Call Parameters

None.

Return Codes

The Reserve function returns one of the following to the REXX variable *rc*:

Code	Description
0	OK, keyboard now locked
1	Not connected to a session
5	Not locked, session in input inhibit state
9	System error

Reset_System

This function resets the system parameters to the default state and disconnects all connected sessions. The system parameters are set initially with the Set_Session_Parms function.

Prerequisite

None.

Syntax

```
HLLAPI( 'Reset_system' )
```

Call Parameters

None.

Return Codes

The Reset_System function returns one of the following to the REXX variable *rc*:

Code	Description
0	OK, reset complete
9	System error

Search_PS

This function searches the currently connected session's PS for the string *string* starting at position *pos*. If SRCHALL has been specified using the Set_Session_Parms function, then *pos* is not used.

Prerequisite

Connect

Syntax

```
HLLAPI( 'Search_PS' , string , pos )
```

Call Parameters

Parameter	Description
string	The string to search for
pos	The starting position for the search

Return Codes

The Search_PS function returns one of the following to the REXX variable *rc*:

Code	Description
0	The string was not found or session not connected
n	Numeric position of string in the connected PS

Search_Field

This function searches the currently connect session's PS for a field containing the string *string*, starting at position *pos*.

Prerequisite

Connect

Syntax

```
HLLAPI( 'Search_field' , string , pos )
```

Call Parameters

Parameter	Description
string	The string to search for
pos	The starting position for the search

Return Codes

The Search_Field function returns one of the following to the REXX variable *rc*:

Code	Description
0	The string was not found or session not connected
n	Numeric position of field containing the specified string in the connected PS

Send_File

This function executes the SEND.EXE program as if it had been entered at the command line.

Prerequisite

None.

Syntax

```
HLLAPI( 'Send_file', string )
```

Call Parameters

Parameter	Description
string	Parameters in an acceptable syntax for the SEND command. The parameters for the SEND command are: `<pc file name> <session id> <host file name> <options>` For more information on file transfer commands and options, see Appendix A.

Return Codes

The Send_File function returns one of the following to the REXX variable *rc*:

Code	Description
2	Parameter error
3	File transfer complete
4	File transfer complete, records segmented
9	System error

Code	Description
27	Terminated by ^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

Sendkey

This function sends the keys as defined by the contents of *string* to the currently connected session. Up to a total of 255 keys may be sent at a single time.To send special control keys, use the compound character coding scheme (as described in Appendix B). It is possible to represent all necessary keystrokes, including 3270 function keys in ASCII, by using an escape character (default value is @) followed by the appropriate key code. Appendix B, “3270 PC Keyboard Mnemonics,” provides a complete list of these key codes.

Prerequisite

Connect

Syntax

```
HLLAPI( 'Sendkey' , string )
```

Call Parameters

Parameter	Description
string	A string of maximum 255 characters (keystrokes) to be sent to the host PS

Return Codes

The Sendkey function returns one of the following to the variable *rc*:

Code	Description
0	OK, keystrokes sent
1	Not connected to a session
4	Session busy, all keystrokes not sent
5	Session is in input inhibit state
6	Bad keystroke mnemonic

Set_Cursor_Pos

This function positions the cursor at position *pos* in the currently connected session.

Prerequisite

Connect

Syntax

```
HLLAPI( 'Set_cursor_pos', pos )
```

Call Parameters

Parameter	Description
pos	Screen position at which to position the cursor

Return Codes

The Set_Cursor_Pos function returns one of the following to the REXX variable *rc*:

Code	Description
0	OK, cursor position changed
1	Not connected to a session

Start_Host_Notify

This function starts trapping of host session screen update events. The session specified by *session_id* is monitored for changes as requested by option.

Prerequisite

None.

Syntax

```
HLLAPI( ’Start_host_notify’ , session_id , option )
```

Call Parameters

The valid values for option are as follows:

Parameter	Description
session_id	The single-character short name of the session. If necessary, use the Query_Sessions call to determine the short names of the sessions that are configured on the system, or check the configuration in the Executive.
option	’P’ Screen changes only ‘O’ OIA changes only ‘B’ Both OIA and screen changes

Return Codes

The Start_Host_Notify function returns one of the following to the REXX variable *rc*:

Code	Description
0	OK, update notify started
1	Not connected
2	Parameter error
9	System error

Set_Session_Parms

This sets the current session parameters from *string*.

Prerequisite

None.

Syntax

```
HLLAPI( 'Set_session_parms' , string )
```

Call Parameters

The session parameter values are ASCII strings as described in the table over the following pages (‘*’ denotes the default value for this setting):

Parameter	Description
‘ATTRB’	On Read_PS calls, pass back ONLY the attributes of the read string
*‘NOATTRB’	Pass back only the ASCII text in the read string
‘EAB’	Pass back extended attributes AND text in the read string
*‘NOEAB’	Pass back only text in the read string

Parameter	Description
'XLATE'	Translate EABs to PC attribute values in the read string
*'NOXLATE'	Do not translate EABs to PC attribute values in the read string
'CONPHYS'	Connect to the physical session, and if required, jump to it
*'CONLOG'	Connect to the logical session requested, do not jump to it
'ESC=*n*'	Set the keystroke mnemonic character to some character *n*, default '@'
*'SRCHALL'	Search the entire PS when doing searches
'SRCHFROM'	Search ONLY from the specified position to the end of the PS when doing searches
*'SRCHFRWD'	Search from the beginning toward end of the PS (forward direction)
'SRCHBKWD'	Search from the end of the PS toward beginning (backward direction)
*'AUTORESET'	Will attempt to reset all inhibit conditions automatically on Sendkey requests
'NORESET'	No reset will be attempted automatically

Parameter	Description
*'TWAIT'	WAIT will wait up to one minute for XCLOCK or XSYSTEM to clear before timing out
'LWAIT'	WAIT will wait until XCLOCK or XSYSTEM clear before returning. Use this option with caution as control will NOT return until the host is available
'NWAIT'	No wait, return status on any WAIT calls
*'FPAUSE'	Full pause will take place regardless of any Start_Host_Notify in effect
'IPAUSE'	Interruptible pause. If a pause is in progress, and a host event occurs while Start_Host_Notify is in effect, this pause will end.
*'NOQUIET'	Display SEND and RECEIVE messages
'QUIET'	So not display file-transfer messages

Call Parameters (cont.)

Parameter	Description	
*'TROFF'	Do not generate EHLLAPI trace data	
'TRON'	Generate EHLLAPI trace data.	
*'TIMEOUT=0'	Do not issue CTRL+BREAK for file transfer if host is not responding	
'TIMEOUT=*value*'	Issue CTRL+BREAK for SEND/RECEIVE file transfer if host is not responding	
	value	**Timeout in Minutes**
	1	0.5
	2	1.0
	3	1.5
	4	2.0
	5	2.5
	6	3.0
	7	3.5
	8	4.0
	9	4.5
	J	5.0
	K	5.5
	L	6.0
	M	6.5
	N	7.0

Return Codes

The Set_Session_Parms function returns one of the following to the REXX variable *rc*:

Code	Description
0	OK, session parameters set
2	One or more parameters are incorrect
9	System error

Set_Window_Status

This function changes the PM window status of a session.

Prerequisite

Connect_PM.

Syntax

```
HLLAPI( ’Set_window_status’ , session_id , option  [, num1 |
option1, num2 ])
```

Call Parameters

The valid values for option are as follows:

Parameter	Description
session-id	The single-character short name of the session.

Call Parameters (cont.)

Parameter	Description
Option	'V' Make the window visible. 'I' Make the window invisible. 'A' Make the window active. 'D' Make the window inactive. 'R' Restore the window from maximized or minimized state. 'Z' Change the window placement based on the first character of option1: **Top** Move the emulation window to the foreground. **Bottom** Move the emulation window to the background. 'X' Maximize the window. 'N' Minimize the window. 'M' Where num1 and num2 represent the decimal position of the upper left corner of the new position. 'S' Where num1 and num2 represent the decimal width and height of the new window.

The num1 and num2 parameters are used only for the Move and Size options and the option1 parameter is used for the Zorder option.

Return Codes

The Set_Windows_Status function returns one of the following to the REXX variable *rc*:

Code	Description
0	The Set_Window_Status function was successful.
1	Your program is not currently connected to the host session.
9	A system error occurred.
12	The session stopped.

Start_Close_Intercept

This function intercepts close requests for the host session.

Prerequisite

None.

Syntax

```
HLLAPI( 'Start_close_intercept' , session_id , option )
```

Call Parameters

The valid values for option are as follows:

Parameter	Description
session-id	The single-character short name of the session.

Return Codes

The Start_Close_Intercept function returns one of the following to the REXX variable *rc*:

Code	Description
0	The Start_Close_Intercept function was successful.
1	An incorrect host presentation space was specified.
9	A system error occurred.
10	The function was not supported by the emulation program.

Start_Host_Notify

This function starts trapping of host session screen update events. The session specified by *session_id* is monitored for changes as requested by option.

Prerequisite

None.

Syntax

```
HLLAPI( 'Start_host_notify' , session_id , option )
```

Call Parameters

The valid values for option are as follows:

Parameter	**Description**
session_id	The single-character short name of the session. If necessary, use the Query_Sessions call to determine the short names of the sessions that are configured on the system, or check the configuration in the Executive.
option	'P' Screen changes only 'O' OIA changes only 'B' Both OIA and screen changes

Return Codes

The Start_Host_Notify function returns one of the following to the REXX variable *rc*:

Code	Description
0	OK, update notify started
1	Not connected
2	Parameter error
9	System error

Start_Keystroke_Intercept

This function begins interception and buffering of all user keystrokes to the session specified by *session_id*. Typically, these keystrokes are then processed by the Get_key and Intercept_status operations.

Prerequisite

None.

Syntax

```
HLLAPI( 'Start_keystroke_intercept' , session_id , option )
```

Call Parameters

Parameter	Description
session_id	The single-character short name of the session. If necessary, use the Query_Sessions call to determine the short names of the sessions that are configured on the system, or check the configuration in the Executive.
option	'D' AID keys only 'L' All keystrokes

Return Codes

Code	Description
0	OK, monitoring started
1	Not connected to a session
2	Option was not correct
4	Session busy and cannot be connected
9	System error

Stop_Close_Intercept

This function allows the application to turn off the Start_Close_Intercept function. After the Stop_Close_Intercept function is issued, subsequent close requests are accepted for the session specified by session_id.

Prerequisite

Start_Close_Intercept

Syntax

```
HLLAPI( 'Stop_close_intercept', session_id )
```

Call Parameters

Parameter	Description
session_id	The single-character short name of the session.

Return Codes

Code	Description
0	The Stop_Close_Intercept function was successful.
1	An incorrect host presentation space was specified.
8	No previous Start_Close_Intercept function was issued.
9	A system error occurred.
12	The session was stopped.

Stop_Host_Notify

This function ends trapping of host session session_id screen update events started by a preceding Start_Host_Notify call.

Prerequisite

Start_Host_Notify

Syntax

```
HLLAPI( 'Stop_host_notify', session_id )
```

Call Parameters

Parameter	Description
session_id	The single-character short name of the session. If necessary, use the Query_Sessions call to determine the short names of the sessions that are configured on the system, or check the configuration in the Executive.

Return Codes

Code	Description
0	OK, intercept ended
1	Not connected to a session
8	No prior Start_Nost_Notify was done
9	System error

Stop_Keystroke_Intercept

This function ends trapping of keystrokes for *session_id* started by a preceding Start_Keystroke_Intercept call.

Prerequisite

Start_Keystroke_Intercept

Syntax

```
HLLAPI( 'Stop_keystroke_intercept', session_id )
```

Call Parameters

Parameter	Description
session_id	The single-character short name of the session. If necessary, use the Query_Sessions call to determine the short names of the sessions that are configured on the system, or check the configuration in the Executive.

Return Codes

Code	Description
0	OK, intercept ended
1	Not connected to a session
8	No prior Start_Keystroke_Intercept was done
9	System error

Wait

This function checks the status of the currently connected session. The setting (set by the Set_Session_Parms call) of TWAIT, NWAIT or LWAIT affects how long this function will wait.

Prerequisite

Connect

Syntax

```
HLLAPI( 'Wait' )
```

Call Parameters

None.

Return Codes

Code	Description
0	Keyboard unlocked and ready for keystrokes
1	Not connected to a session
4	Wait timed out while still in XCLOCK or XSYSTEM state
5	Keyboard locked
9	System error

Sample Program

The following sample program illustrates the way that the REXX EHLLAPI functions are registered to the REXX interpreter, and how the function calls are coded within the application.

```
/* LOG.CMD */

/* Depending on Terminal type moves cursor and does a logon to a VM Host */

/* then checks the current host time */

if RxFuncQuery(’hllapi’) then

        call RxFuncAdd ’hllapi’,’saahlapi’,’hllapisrv’

        say ’Which Session would you like to connect to?’

        pull session

        rc =hllapi(Connect,session)

        if rc <> 0 then do

                say "Connect Failed, program will terminate"

                say hllapi(Disconnect)

                exit

        end

        rc=hllapi(Query_Session_Status,session)

        x=substr(rc,12,2)

        x=c2d(reverse(x))

        y=substr(rc,14,2)

        y=c2d(reverse(y))

        if (x == 24 & y == 80) then

                z=2

        else if (x == 32 & y == 80) then

                z=3

        else if (x == 43 & y == 80) then

                z=4

        else if (x == 27 & y == 132) then
        z=5
else do
        say "Error: Query Session Status, program terminated"
        say hllapi(Disconnect)
        exit
end
say "This Session is a Model " z " Terminal"
say "Would you like to log in? (Y/N)"
pull answer
if answer == "Y" | answer == "y" then do
        if (z == 2)  then
                say hllapi(Set_Cursor_Pos,1536)
        else if (z == 3) then
                say hllapi(Set_Cursor_Pos,2176)
        else if (z == 4) then
                say hllapi(Set_Cursor_Pos,3056)
        else if (z == 5) then
                say hllapi(Set_Cursor_Pos,2920)
        say "What is your USERID?"
        pull user
        say "What is your PASSWORD?"
        say "Warning it will not be hidden?"
        pull password
        say hllapi("SendKey",user)
        x=0
        z=(8-length(user))
        do while x <> z
                rc=hllapi('SendKey',' ')
                x=x+1
        end
        say hllapi('SendKey',password)
        say hllapi('SendKey','@E')
        say hllapi('Set_Session_Parms',SRCHALL SRCHFRWD)
        do until rc > 0
           rc= hllapi('Search_PS','Ready',1)
        end
        rc=hllapi('Sendkey', 'CP Query Time @E')
        rc=hllapi('Wait')
        pos=hllapi('Search_ps','TIME IS ',1)
        if pos=0 then do
           say 'Host could not process QUERY TIME command.'
           exit
        end
        else do
           time=hllapi('Copy_ps_to_str', pos + Length('TIME IS '), 8)
           say "Current Host time is: " time
        end
end
rc=hllapi(Disconnect)
exit
```

Appendix B: 3270-PC Keyboard Mnemonics

The following key codes allow you to represent 3270-PC special function keys in your calling data strings. You can use these codes with the Sendkey function to specify the keystrokes you want to send, as well as with the Get_key function which receives the keystrokes sent through using Sendkey.

These codes rely on ASCII characters to represent the special function keys of the 3270-PC. For example, to send the keystroke Alt PF1, you would code "@A@1" with @A representing the Alt key. And to represent a shifted Jump keystroke, you would code "@S@J" with @S representing the shift.

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 "@" 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 "@@."

Each of the codes on the following chart should be preceded by the Escape character which defaults to the "@" sign:

A	Alt	J	Jump	S	Shift
B	Backtab	K	Copy	T	Tab
C	Clear	L	Cursor Left	U	Cursor Up
D	Delete	M	Enlarge	V	Cursor Down
E	Enter	N	New Line	W	Not Used
F	Erase EOF	O	Not Used	X	Reserved
G	Not Used	P	Print	Y	Not Used
H	Help	Q	Quit	Z	Cursor Right
I	Insert	R	Reset		

0	Home	a	PF10	k	PF20
1	PF1	b	PF11	l	PF21
2	PF2	c	PF12	m	PF22
3	PF3	d	PF13	n	PF23
4	PF4	e	PF14	o	PF24
5	PF5	f	PF15	x	PA1
6	PF6	g	PF16	y	PA2
7	PF7	h	PF17	z	PA3
8	PF8	i	PF18		
9	PF9	j	PF19		

Appendix A: File Transfer Commands

This appendix lists the file transfer commands for REXX EHLLAPI used with the Send_file function and the Receive_file function.

File Transfer Commands

The following commands must be used with the REXX EHLLAPI command parameters in the Send_file function and the Receive_file function.

The first three components are required, the final two are optional, and all must appear in the order shown. The components of the strings are as follows:

(See also the CMS and TSO examples given below.)

```
<pc filename> <session id> <host file name> <options>
<buffered/BUFSIZE>
```

<pc filename>

[d:][path]filename[.ext]

A path and filename for the source or destination of the transfer.

d:	Optional drive specifier for the file to be transferred. If not specified, the default drive is used.
path	Optional directory path of the PC file to be transferred.
filename	The PC filename to be transferred.
.ext	The PC filename extension, if any.

<session id>

session id is the session short name plus a colon, for example, “c:” for Session C.

<host file name>

The host file name in the format suitable for the host operating system and host file transfer application. Note that the file names are different for TSO and CMS. Refer to example on page 69.

<options>

append	Lets you append the transferred file to the end of an existing file. The append option overrides any specified values for the “lrecl” and “recfm” options.
	Note: If “append” is not used when the filename already exists on the mainframe and on the PC, the existing file will be replaced. “Append” is not valid with PDS (partitioned data set).
ascii	This option converts EBCDIC to ASCII (codepage 1004) before it is transferred to your PC. ASCII mode is recommended if the file is to be PC-readable.
codepage=nnn	This option enables you to change the translation code page referred to when you use the *ascii* option. *nnn* is the number of the code page that you want the transfer program to use. For example, `codepage=437`.
crlf	This option deletes carriage return/line feed characters. Usually, alphanumeric data contains a “crlf” character at the end of each line to denote the end of each record for the PC files. Deleting these characters using the “crlf” option lets you read the file more easily on the host.
lrecl n	(Send only) Specifies the logical record length of the mainframe file, where n is the number of characters in each record. Default is 80 for new files. If you are replacing a file, the default is the lrecl of the existing file. If you have specified the “append” option, “lrecl” is ignored. When working with variable-length records, n is the maximum length record the mainframe will accept.
recfm f	(Send only) Specifies that the mainframe file will contain fixed-length records. Records are padded with space characters if they are shorter than that specified with “lrecl” or the default length. See following note.
recfm v	(Send only) Specifies that the file will contain variable length records. See following note.

Note: If you use the "append" option, the "recfm" option is ignored. If you do not specify "recfm," the default (f) is used unless you specify "crlf." When you specify "crlf," the default is v (variable length). For existing files, the default is the "recfm" of the existing file.

Use of the "recfm v" option without the "crlf" option is recommended when padding of mainframe records or deletion of trailing spaces must not occur.

<buffered/BUFSIZE>

This option refers to two types of optional send or receive parameters.

- **-B** to force the transfer to be a buffered file transfer. The default for a DFT session is structured field file transfer. CUT sessions support only buffered file transfer.
- **BUFSIZE=** sets the size of structured field file transfer buffers, in K bytes. The possible values are the following:

  "BUFSIZE=2" (default)

  "BUFSIZE=4"

  "BUFSIZE=8"

  "BUFSIZE=32"

Examples

The components are illustrated in the examples below. The first shows a structured field SEND transfer in the CMS system with 32K structured fields. The second shows a buffered RECEIVE transfer in the TSO system.

CMS: Sending (using REXX EHLLAPI Send_file)

```
c:\saledata\salejuly.txt e: julysale data (ascii crlf BUFSIZE=32
```

TSO: Receiving (using REXX EHLLAPI Receive_File)

```
c:\orders\orderdat.doc b: orderdata.text(mem) ascii append crlf -B
```