Personal Communications/3270
for DOS (Entry-Level Mode)

IBM

Programmer's Guide

Version 4.1

SC31-8267-00

Personal Communications/3270
for DOS (Entry-Level Mode)

IBM

Programmer's Guide

Version 4.1

SC31-8267-00

Note

Before using this information and the product it supports, be sure to read the general information under “Notices” on page ix.

First Edition (March 1996)

This edition applies to Version 4.1 of Personal Communications for DOS. It applies to all subsequent releases and modifications until otherwise indicated in new editions or technical newsletters. Make sure you are using the correct edition for the level of the product.

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address below.

IBM welcomes your comments. A form for readers' comments may be provided at the back of this publication. If it has been removed,you may send your comments to the following address:

International Business Machines Corporation
Department CGMD
P.O. Box 12195
Research Triangle Park, North Carolina
27709-2195

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

© **Copyright International Business Machines Corporation 1996. All rights reserved.**
Note to U.S. Government Users — Documentation related to restricted rights — Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

Contents

EEHLLAPI and AHLLAPI Functions by Function Number vii

Notices ix
Trademarks and Service Marks x

About This Book xi
How This Book Is Organized xii
Who Should Read This Book xii
Requirements xiii
  Hardware Requirements xiii
  Software Requirements xiii
Compatibility with Other Emulation Products xiii
Host Requirements xiv
Related Publications xiv

Chapter 1. Introduction 1
Entry HLLAPI Overview 1
What You Need to Get Started 2
  The Base Code (PC3270.COM) 2
  The Base Code (ASC3270.COM) 2
  The Resident Module (EEHLLAPI.EXE) 2
  The Resident Module (AHLLAPI.EXE) 3
  Language Interface Modules (LIMs) 3
  Linking Your Entry HLLAPI Program with Its LIM 3
  Function Calls 3

Chapter 2. Loading Entry HLLAPI and Running the Program Sampler . . . 5
Loading Entry HLLAPI for CUT Connection 5
Loading Entry HLLAPI for Home3270 5
Installing and Running the Program Sampler 6

Chapter 3. EEHLLAPI Functions 7
Page Layout Conventions 7
  Call Parameters 7
  Return Parameters 8
  Notes on Using This Function 8
Connect Presentation Space (1) 9
  Call Parameters 9
  Return Parameters 9
  Notes on Using This Function 10
Convert Position or RowCol (99) 11
  Call Parameters 11
  Return Parameters 11
Copy Field to String (34) 12
  Call Parameters 12
  Return Parameters 12
Copy OIA (13) 13
  Call Parameters 13
  Return Parameters 13
Copy Presentation Space (5) 14

© Copyright IBM Corp. 1996

Call Parameters . . . 14
Return Parameters . . . 14
Notes on Using This Function . . . 14
Copy Presentation Space to String (8) . . . 15
Call Parameters . . . 15
Return Parameters . . . 16
Copy String to Field (33) . . . 17
Call Parameters . . . 17
Return Parameters . . . 17
Copy String to Presentation Space (15) . . . 18
Call Parameters . . . 18
Return Parameters . . . 18
Notes on Using This Function . . . 18
Disconnect Presentation Space (2) . . . 19
Call Parameters . . . 19
Return Parameters . . . 19
Notes on Using This Function . . . 19
Find Field Length (32) . . . 20
Call Parameters . . . 20
Return Parameters . . . 20
Find Field Position (31) . . . 21
Call Parameters . . . 21
Return Parameters . . . 21
Pause (18) . . . 22
Call Parameters . . . 22
Return Parameters . . . 22
Notes on Using This Function . . . 22
Query Cursor Location (7) . . . 23
Call Parameters . . . 23
Return Parameters . . . 23
Query Field Attribute (14) . . . 24
Call Parameters . . . 24
Return Parameters . . . 24
Query Host Update (24) . . . 25
Call Parameters . . . 25
Return Parameters . . . 25
Query Session Status (22) . . . 26
Call Parameters . . . 26
Return Parameters . . . 26
Query Sessions (10) . . . 27
Call Parameters . . . 27
Return Parameters . . . 27
Notes on Using This Function . . . 27
Query System (20) . . . 28
Call Parameters . . . 28
Return Parameters . . . 28
Notes on Using This Function . . . 29
Receive File (91) . . . 30
Call Parameters . . . 30
Return Parameters . . . 31
Notes on Using This Function . . . 32
Considerations for Using Functions 90 and 91 . . . 33
Release (12) . . . 34
Call Parameters . . . 34

Return Parameters . . . 34
Notes on Using This Function . . . 34
Reserve (11) . . . 35
Call Parameters . . . 35
Return Parameters . . . 35
Notes on Using This Function . . . 35
Reset System (21) . . . 36
Call Parameters . . . 36
Return Parameters . . . 36
Search Field (30) . . . 37
Call Parameters . . . 37
Return Parameters . . . 37
Notes on Using This Function . . . 38
Search Presentation Space (6) . . . 39
Call Parameters . . . 39
Return Parameters . . . 39
Notes on Using This Function . . . 39
Send File (90) . . . 41
Call Parameters . . . 41
Return Parameters . . . 42
Notes on Using This Function . . . 43
Send Key (3) . . . 45
Call Parameters . . . 45
Return Parameters . . . 45
Notes on Using This Function . . . 45
Keyboard Mnemonics . . . 46
AID Key Mnemonics . . . 47
Set Session Parameters (9) . . . 48
Call Parameters . . . 48
Return Parameters . . . 51
Start Host Notification (23) . . . 52
Call Parameters . . . 52
Return Parameters . . . 53
Notes on Using This Function . . . 53
Stop Host Notification (25) . . . 54
Call Parameters . . . 54
Return Parameters . . . 54
Notes on Using This Function . . . 54
Storage Manager (17) . . . 55
Get Storage . . . 56
Call Parameters . . . 56
Return Parameters . . . 56
Notes on Using This Function . . . 56
Free Storage . . . 57
Call Parameters . . . 57
Return Parameters . . . 57
Free All Storage . . . 58
Call Parameters . . . 58
Return Parameters . . . 58
Wait (4) . . . 59
Call Parameters . . . 59
Return Parameters . . . 59
Notes on Using This Function . . . 59

Chapter 4. AHLLAPI Functions 61
Functional Differences between AHLLAPI and EEHLLAPI 61
Copy Field to String (34) 61
Copy OIA (13) 61
Copy Presentation Space (5) 62
Copy Presentation Space to String (8) 62
Copy String to Field (33) 62
Copy String to Presentation Space (15) 62
Disconnect Presentation Space (2) 62
Find Field Position (31) 63
Find Field Length (32) 63
Query Field Attribute (14) 63
Query System (20) 63
Search Field (30) 64
Send Key (3) 64
Set Session Parameters (9) 65
Additional AHLLAPI Functions 66
Drop Session DTR (82) 66
Query Emulator Host Connection (80) 66
Raise Session DTR (81) 67

Chapter 5. Compiling and Running Your EEHLLAPI Application Program 69
Linking Your EEHLLAPI Application Program with the Appropriate LIM 69
Using BASIC 69
Interpretive BASIC 69
Compiled BASIC 71
Using COBOL 72
Using Pascal 73
Using IBM C 76
Using 8088 Assembler 76
Running Your EEHLLAPI Application Program 77
Using Trace to Help You Debug Your Program 77

Appendix A. EEHLLAPI Messages 79

Appendix B. Writing Your Own Language Interface Module 81
Functions a LIM Can Perform 83
Revised Call Support for the Personal Computer 84
Details of Writing Your Own Language Interface Module 85

Appendix C. OIA Image and Bit Group Information 87
OIA Image Group (Positions 2 through 81) 87
OIA Group Indicator Meanings (Positions 82-103) 88

Appendix D. Sample Programs 89
BASIC Sample Program 89
COBOL Sample Program 95
Pascal Sample Program 101
“C” Sample Program 109

Appendix E. Alternate Code Page Support 115

Index 117

EEHLLAPI and AHLLAPI Functions by Function Number

The following list shows the page numbers where EEHLLAPI and AHLLAPI functions are explained in this book.

1 "Connect Presentation Space (1)" on page 9
2 "Disconnect Presentation Space (2)" on page 19 and 62
3 "Send Key (3)" on page 45 and 64
4 "Wait (4)" on page 59
5 "Copy Presentation Space (5)" on page 14 and 62
6 "Search Presentation Space (6)" on page 39
7 "Query Cursor Location (7)" on page 23
8 "Copy Presentation Space to String (8)" on page 15 and 62
9 "Set Session Parameters (9)" on page 48 and 65
10 "Query Sessions (10)" on page 27
11 "Reserve (11)" on page 35
12 "Release (12)" on page 34
13 "Copy OIA (13)" on page 13 and 61
14 "Query Field Attribute (14)" on page 24 and 63
15 "Copy String to Presentation Space (15)" on page 18 and 62
17 "Storage Manager (17)" on page 55
18 "Pause (18)" on page 22
20 "Query System (20)" on page 28 and 63
21 "Reset System (21)" on page 36
22 "Query Session Status (22)" on page 26
23 "Start Host Notification (23)" on page 52
24 "Query Host Update (24)" on page 25
25 "Stop Host Notification (25)" on page 54
30 "Search Field (30)" on page 37 and 64
31 "Find Field Position (31)" on page 21 and 63
32 "Find Field Length (32)" on page 20 and 63
33 "Copy String to Field (33)" on page 17 and 62
34 "Copy Field to String (34)" on page 12 and 61
80 "Query Emulator Host Connection (80)" on page 66
81 "Raise Session DTR (81)" on page 67
82 "Drop Session DTR (82)" on page 66
90 "Send File (90)" on page 41
91 "Receive File (91)" on page 30
99 "Convert Position or RowCol (99)" on page 11

© Copyright IBM Corp. 1996

Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program or service is not intended to state or imply that only that IBM product, program, or service may be used. Subject to IBM's valid intellectual property or other legally protectable rights, any functionally equivalent product, program, or service may be used instead of the IBM product, program, or service. The evaluation and verification of operation in conjunction with other products, programs, or services, except those expressly designated by IBM, are the responsibility of the user.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, New York 10594, USA.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

Site Counsel
IBM Corporation
P.O. Box 12195
3039 Cornwallis
Research Triangle Park, NC 27709-2195
USA

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement.

This document is not intended for production use and is furnished as is without any warranty of any kind, and all warranties are hereby disclaimed including the warranties of merchantability and fitness for a particular purpose.

© Copyright IBM Corp. 1996

Trademarks and Service Marks

The following terms, denoted by an asterisk (*) in this publication, are trademarks of the IBM Corporation in the United States or other countries or both:

AT
C/2
COBOL/2
IBM
Macro Assembler/2
Personal Computer AT
PS/2

Windows is a trademark of Microsoft Corporation.

PC Direct is a trademark of Ziff Communications Company and is used by IBM Corporation under license.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

Other company, product, and service names, which may be denoted by a double asterisk (**), may be trademarks or service marks of others.

About This Book

This book provides programming information you will need to use the Personal Communications (hereafter called *PC/3270*) for Entry-Level DOS Mode, High-Level Language Application Program Interface (hereafter called *Entry HLLAPI*).

Entry HLLAPI consists of the following two APIs:

EEHLLAPI for CUT connection

AHLLAPI for Home3270

Entry HLLAPI gives users and application programmers a set of functions that can be called from an application program running in a personal computer (PC) workstation session to access the host presentation space.

© Copyright IBM Corp. 1996

How This Book Is Organized

Chapter 1, “Introduction” provides an overview of Entry HLLAPI and discusses parts of the interface.

Chapter 2, “Loading Entry HLLAPI and Running the Program Sampler” describes how to load the resident portion of Entry HLLAPI and run the Entry HLLAPI program sampler.

Chapter 3, “EEHLLAPI Functions” describes each function in detail. These functions are listed in alphabetic order.

Chapter 4, “AHLLAPI Functions” describes additional information for AHLLAPI functions based on EEHLLAPI functions. These functions are listed in alphabetic order.

Chapter 5, “Compiling and Running Your EEHLLAPI Application Program” provides information about linking your Entry HLLAPI program with the appropriate Language Interface Module (LIM), running your program, and using Trace.

Appendix A, “EEHLLAPI Messages” explains the messages you might see while using Entry HLLAPI.

Appendix B, “Writing Your Own Language Interface Module” describes how you can write a Language Interface Module for any computer language you want to use beyond those supported by LIMs on your Entry-Level DOS Mode diskette.

Appendix C, “OIA Image and Bit Group Information” explains information needed to interpret the returned data string under the **Copy OIA** (13) function.

Appendix D, “Sample Programs” contains code samples.

Appendix E, “Alternate Code Page Support” lists the code pages supported for each country.

Who Should Read This Book

This book is intended for PC users and programmers who want to write application programs that use the services of Entry HLLAPI.

A working knowledge of the Personal Computer and IBM DOS is assumed. If you want more information about the Personal Computer, refer to the list of IBM Personal Computer publications in “Related Publications” on page xiv.

This book assumes you are familiar with the language and compiler you will be using. If you need more information on how to write, compile, or link-edit programs, refer to the appropriate reference books for the specific language.

Requirements

Before you can use Entry HLLAPI, you must understand the following hardware, software, and host computer requirements.

Hardware Requirements

Refer to *Personal Communications Version 4.1 Up and Running for DOS (Entry-Level Mode)*.

Software Requirements

The following software guidelines should help you use Entry HLLAPI:

DOS Level Version 3.3 or higher.

Personal Communications/3270 for Entry-Level DOS Mode Program.

You must write your Entry HLLAPI program in one of the following high-level languages (or its equivalent):

- IBM BASIC (either Compiled or Interpretive)
- IBM BASIC/2 (Compiler)
- IBM C
- IBM C/2
- IBM COBOL
- IBM COBOL/2
- IBM Macro Assembler
- IBM Macro Assembler/2
- IBM Pascal
- IBM Pascal/2

Host graphics are not supported by Entry HLLAPI.

Note: For the latest information on available hardware and software that can be used with this product, see your authorized IBM Personal Computer dealer or local IBM representative.

Compatibility with Other Emulation Products

The EEHLLAPI provides a subset of the programming functions offered by the IBM 3270 Personal Computer High-Level Language Application Program Interface. Programs written using EEHLLAPI can be run unchanged on a properly configured PC/3270. This compatibility provides for easy migration of programs from the PC/3270 for Entry-Level DOS Mode to the 3270 PC. Refer to the latest level of *IBM 3270 Personal Computer High-Level Language Application Program Interface, Programming Guide*, for more details.

Host Requirements

There are no additional requirements other than those for the PC/3270 for Entry-Level DOS Mode.

Related Publications

Although it is assumed that you are familiar with the operation of the Personal Computer and the related languages, you might want to refer to the publications listed below:

Personal Communications Version 4.1 Up and Running for DOS (Entry-Level Mode), SC31-8263

IBM Disk Operating System User's Guide

IBM Disk Operating System Technical Reference

IBM Personal Computer Language Series: C Compiler Compile, Link and Run

IBM PC BASIC Manual

IBM PC BASIC Compiler Manual

IBM PC COBOL Reference

IBM PC Pascal Manual

IBM PC Macro Assembler Manual

Chapter 1. Introduction

Entry HLLAPI Overview

The Entry HLLAPI lets you write and use personal computer programs in IBM* BASIC, Pascal, COBOL, C, or Macro Assembler to interact with your host session. It can increase productivity for experienced users and provide a shorter learning curve for inexperienced users. Entry HLLAPI:

1. Improves overall ease of use by:

   Automating repetitive tasks

   Masking complex applications from the user

   Consolidating several complicated tasks into one simple task.

2. Simplifies existing host applications.

3. Provides unattended operation through a **program operator** that monitors tasks without human intervention. This program operator can:

   Monitor events that are serial in nature (for example, data center applications)
   Automate console operation
   Monitor response time and availability
   Do stress testing.

4. Allows you to create **composite screen** applications using input from the host. The input for a composition screen is formatted and presented within a PC presentation space.

5. Allows you to write programs that divide the work between host and PC sessions. Entry HLLAPI will let you write and use an application to access files and programs in either the host or the PC session. The more trivial tasks (file editing, simple graphics, and so on) can be done in the PC session. The more powerful host can do database searches, updates, and retrievals when needed. The terminal operator does not even need to know whether a PC or host application is doing the work.

In other words, you can use Entry HLLAPI with the functions provided by the PC/3270 Entry-Level DOS mode to enhance interaction between a workstation application program and a host session, and to simplify your programming tasks.

© Copyright IBM Corp. 1996

What You Need to Get Started

To use Entry HLLAPI, you will need the following things:

1. An application program that you or someone else has written using IBM BASIC, COBOL, Pascal, C, or Macro Assembler to call on the Entry HLLAPI functions. If you do not have one yet, you can try out the Program Sampler that is provided on the Entry-Level DOS Mode diskette. (See Chapter 2, "Loading Entry HLLAPI and Running the Program Sampler" for more information.)

   The other files that you will need are on your Entry-Level DOS Mode diskette.

2. PC327 .COM: This is the base Entry-Level DOS Mode code.
3. ASC327 .COM: Additionally required when Home3270 is used.
4. EEHLLAPI.EXE: This is the EEHLLAPI interface module. It lets your program interact with the base code.
5. AHLLAPI.EXE: Required instead of EEHLLAPI.EXE when Home3270 is used.
6. Except for interpretive BASIC, and assembler language programs, you will need one of the following Language Interface Modules (LIMs):

   HLLC_S.OBJ: LIM for IBM "C" Language (small size)

   HLLC_M.OBJ: LIM for IBM "C" Language (medium size)

   HLLC_L.OBJ: LIM for IBM "C" Language (large size)

   HLLCBAS.OBJ: LIM for Compiled IBM BASIC

   HLLCOB.OBJ: LIM for IBM COBOL

   HLLPAS.OBJ: LIM for IBM Pascal.

The LIM you choose depends on the programming language you intend to use. For example, if you were programming in COBOL, you would need the HLLCOB.OBJ file. (This book will explain how to use LIMs and how to write your own LIMs later on.)

The Base Code (PC3270.COM)

This is the code that provides the base emulation function for the CUT connection on your workstation. It must be loaded before loading EEHLLAPI.EXE.

The Base Code (ASC3270.COM)

This is the code that provides the additional base emulation function for Home3270 on your workstation. It must be loaded after loading PC3270.COM and before loading AHLLAPI.EXE.

The Resident Module (EEHLLAPI.EXE)

The resident module for EEHLLAPI is called EEHLLAPI.EXE. It is required for the CUT connection. It must be loaded before you can use EEHLLAPI and will remain in memory as an extension of DOS.

The Resident Module (AHLLAPI.EXE)

The resident module for AHLLAPI is AHLLAPI.EXE. It is required for Home3270 and must be loaded instead of EEHLLAPI.EXE. It must be loaded before you use AHLLAPI and will remain in memory as an extension of DOS.

Language Interface Modules (LIMs)

LIMs are the "bridges" between your Entry HLLAPI application program and EEHLLAPI.EXE or AHLLAPI.EXE. LIMs are provided because each implementation of a programming language has its own unique methods of storing data and calling subroutines. LIMs allow Entry HLLAPI to support multiple high-level languages.

LIMs are provided for Compiled IBM BASIC, IBM COBOL, IBM C, and IBM Pascal. No LIM is required for Interpretive BASIC. Instead, you must access a special Interpretive BASIC LIM that is available whenever Entry HLLAPI is loaded. Refer to Chapter 5, "Compiling and Running Your EEHLLAPI Application Program" for details.

Programs written using the Macro Assembler can invoke Entry HLLAPI directly. If you prefer to use a language other than those mentioned above, refer to Appendix B, "Writing Your Own Language Interface Module."

Linking Your Entry HLLAPI Program with Its LIM

When you are ready to run your Entry HLLAPI application program, you can make the appropriate LIM a permanent part of your program in this manner:

If you are using a compiled language (such as COBOL or Compiled BASIC), you call the appropriate language interface as an external subroutine. When you link-edit your program (using the DOS **link** command), the appropriate LIM will be merged with your Entry HLLAPI program. Refer to Chapter 5, "Compiling and Running Your EEHLLAPI Application Program" for details.

Function Calls

Entry HLLAPI is a "function-code"—driven system. This means that each Entry HLLAPI function has a number associated with it. This number is the name you use to call the function from a program that you are writing.

Chapter 2. Loading Entry HLLAPI and Running the Program Sampler

This chapter tells you:

How to load Entry HLLAPI

How to load and run the Program Sampler.

Loading Entry HLLAPI for CUT Connection

To load Entry HLLAPI:

1. Make sure you select Yes for the HLLAPI option in the CUT Configuration panel of INSTALL/CONFIG utility. For details of the INSTALL/CONFIG utility, refer to the *Personal Communications Version 4.1 Up and Running for DOS (Entry-Level Mode)*.
2. Start the configuration BAT file for the CUT connection.

   When the program loads, you will see the following messages:

```
Entry Emulator High-Level Language Application Program Interface 4.

(c) Copyright International Business Machines Corporation 1984, 1994

EHL  1 EEHLLAPI is loaded

EHL  2 EEHLLAPI is ready for use

A>
```

   If you receive any other messages, refer to Appendix A, “EEHLLAPI Messages.”

Loading Entry HLLAPI for Home3270

To load Entry HLLAPI:

1. Make sure you select Yes for the HLLAPI option in the Emulation Parameters panel of the INSTALL/CONFIG utility. For details of the INSTALL/CONFIG utility, refer to the *Personal Communications Version 4.1 Up and Running for DOS (Entry-Level Mode)*.
2. Start the configuration BAT file for Home3270.

© Copyright IBM Corp. 1996

Installing and Running the Program Sampler

After you have successfully loaded Entry HLLAPI, you are ready to use the Entry HLLAPI interface. The Entry HLLAPI diskette contains a Compiled BASIC Program Sampler (EHLSAMP.EXE for CUT connection, AHLSAMP.EXE for Home3270) that lets you experiment with the various functions. The Program Sampler allows you to specify a function and its parameters and observe the results. It also serves as an example of running a program written in BASIC in which the Entry HLLAPI functions have been used. When you get ready to write your Entry HLLAPI application program, you might find it useful to refer to the BASIC sample program in Appendix D, "Sample Programs."

To run the Program Sampler EHLSAMP.EXE:

1. Have your Entry-Level DOS Mode program and EEHLLAPI.EXE loaded into the memory.
2. Be sure that the following files are available to the program sampler (on the same diskette or in the same hard disk directory):

   PC utilities that EHLSAMP uses, such as:

   SEND.COM
   RECEIVE.COM.
3. Place your Entry Level diskette that contains EHLSAMP.EXE into drive A.
4. Type EHLSAMP and press Enter. You will see the Entry HLLAPI Program Sampler panel appear.
5. Sample the Entry HLLAPI functions.

Note: When you use Entry Emulator as Home3270, use AHLSAMP.EXE instead of EHLSAMP.EXE.

It is a good idea to have the description of any function you sample (found in Chapter 3) in front of you as you use the Program Sampler. When you use the Program Sampler, you are using actual Entry HLLAPI functions, not simulations of functions. All the rules about the order in which functions should be called apply when using the Program Sampler. For example, when using an EEHLLAPI program, you must be connected to a session (by means of the **Connect Presentation Space** (1) function) before you can send keys to it (using the **Send Key** (3) function).

If you want to quit a function without completing it, press Ctrl+Break. This will take you back to DOS. To use the EEHLLAPI Program Sampler again, type EHLSAMP and press Enter.

When you are ready to exit the Program Sampler and return to DOS, type **300**, and press Enter from the main menu.

For more information about using Compiled BASIC, refer to the *IBM Compiled BASIC Reference Manual*.

Chapter 3. EEHLLAPI Functions

Page Layout Conventions

All EEHLLAPI function calls are presented in the same format so that you can retrieve the information you need quickly. The format looks like this:

Function Name
Call Parameters
Return Parameters
Notes on Using This Function

Call Parameters

"Call Parameters" defines what parameters you need to define in your program in order to use this function and how these parameters are to be defined.

EEHLLAPI function calls pass the following four parameters in this fixed format:

A function number
A data string
The string's length
The host presentation space position

Function Number, the first parameter, is always required. This position must be filled by a 2-byte (fullword) integer.

Data String, the second parameter, is used in different ways by different functions. In some functions, the data string is a string of characters. In other functions, it is a string of concatenated data items.

String Length, the third parameter, is usually the length of the character string or concatenated list of data items. In special cases this parameter passes information such as buffer size. This position must be filled by a 2-byte (fullword) integer.

Presentation Space (PS) Position, the fourth parameter, is a value associated with the IBM 3278/79 Display Station screen sizes emulated by the PC. This position must be filled by a 2-byte (fullword) integer, and must be a value from 1 through 1920 for the Entry Level. When the PS position is not applicable, this 2-byte integer can be any value.

© Copyright IBM Corp. 1996

Return Parameters

"Return Parameters" defines the information that your program will receive from EEHLLAPI after EEHLLAPI has processed your function call.

EEHLLAPI function calls return requested information in the following format:

Function number
Data string
Data string length or host presentation space position
Return code

Note that not all four parameters will be changed on return for each function.

Function Number, the first parameter, is always returned unchanged.

Data String, the second parameter, returns different information according to function. In some functions, the data string is a string of characters. In other functions, it is a string of concatenated data items.

You must preallocate space for returning data strings in your EEHLLAPI application program.

The third parameter, when sending data to EEHLLAPI, is interpreted as **Data Length**. For certain function calls, EEHLLAPI returns a presentation space position. (This is a numeric value from 1 through 1920).

Return Code, the last parameter, is usually a numeric return code. The **Convert Position or RowCol** (99) function is the exception to this rule. In Function 99, the return code position passes data to the program.

Return codes are explained in detail in the descriptions of the individual functions.

Notes on Using This Function

"Notes On Using This Function" explains any prerequisite function calls that should be issued before using the function under discussion. It also provides technical information about using the function, plus application development tips.

See Appendix D, "Sample Programs" for examples on how the functions are used.

Connect Presentation Space (1)

The **Connect Presentation Space** function establishes a connection between your EEHLLAPI application program and the host presentation space.

Call Parameters

Data String: One-character short name of the host presentation space.[1]

Length: NA (1 is implied).

PS Position: NA

The calling data string can contain:

A one-character host presentation space short name.

For the Entry-Level DOS mode, the default presentation space is “E.”

Note: If you are using this default, you must have an “E” in the calling data string.

Return Parameters

The following return codes are valid:

Return Code	Explanation
0	The **Connect Presentation Space** function was successful; the host presentation space is unlocked and ready for input.
1	An incorrect host presentation space ID.
4	Successful connection was achieved, but the host presentation space is busy.
5	Successful connection was achieved, but the host presentation space is locked (input inhibited).
9	A system error was encountered.
11	This resource is unavailable. The host presentation space is already being used by another system function.

Notes on Using This Function

1. The following functions do not require that you issue a **Connect Presentation Space** function call before using:

   Set Session Parameters (9)
   Query Sessions (10)
   Query System (20)
   Reset System (21)
   Query Session Status (22)
   Query Host Update (24)
   Send File (90)
   Receive File (91)
   Convert Position or RowCol (99)

2. The **Connect Presentation Space** function sets the return code to indicate the status of the attempt and, if successful, the status of the host presentation space.

3. Two parameters under Function 9 affect Connect:

Parameter	Explanation
CONPHYS	During the Connect, jump to the requested presentation space (Do a physical connect).
CONLOG	During the Connect, do not jump to the requested presentation space (Do a logical connect).

Convert Position or RowCol (99)

This function converts the host presentation space positional value into the display row/column coordinates, or vice versa. This function does not change the cursor position.

Call Parameters

Data String: Host Presentation space short name and "P" for convert position, or

Host Presentation space short name and "R" for convert row/column.

Length: Row

PS Position: Column (when specify "R" above) or host presentation position (when "P" above is specified).

Return Parameters

This function is the exception to the rule that the return code position always contains a return code. In this instance, it contains a status code. If you have established a common error-handling routine, this function could return misleading information.

Two pieces of information are returned:

The *Length* parameter, which returns the row number or "0" for incorrect row input.

A status code in the *Return Code* parameter. These status codes are explained below.

Status Code	Explanation
0	Incorrect column or presentation space position was provided.
>0	This is the PS position or column.
9998	An incorrect host presentation space ID was specified, or the host presentation space was never connected.
9999	Character 2 in the data string is not P or R.

Copy Field to String (34)

The **Copy Field to String** function transfers characters on the just-connected presentation space into a string. The string begins at the field's origin delimiter. This position and length information can be found by using the **Find Field Position** (31) function and the **Find Field Length** (32) function. This function can be used with either protected or unprotected fields but only in *a field-formatted host presentation space*.

The string ends when one of these three condition is encountered:

When the end of the field is reached.
When the length of the target string is exceeded.
When the end of the host presentation space is reached.

Call Parameters

Data String: Preallocated target data string.

Length: Length of the target data string.

PS Position: Position of the source field in the host presentation space from which to copy.

Return Parameters

This function returns the requested data string and one of the following return codes:

Return Code	Explanation
0	The **Copy Field to String** function was successful.
1	Your program is not currently connected to the emulated host session.
2	An error was made in specifying parameters.
6	The data to be copied and the target field were not the same size. The data might have been truncated because the string length might have been smaller than the field copied.
7	The host presentation space position is not valid.
9	A system error was encountered.

For Home3270, refer to Chapter 4, “AHLLAPI Functions.”

Copy OIA (13)

The **Copy OIA** function returns the current OIA data from the host-connected presentation space.

Call Parameters

Data String: Target string

Length: Length of the target data string.

PS Position: NA

The OIA data is returned in a data string that must be 103 bytes long. The format of the returned string is as follows:

Byte	Definition
Position 1	The OIA Format Byte for the PC.
Positions 2–81	The OIA image in host code points. Refer to Appendix C, "OIA Image and Bit Group Information" for complete explanations of these positions.
Positions 82–103	The OIA group. Refer to Appendix C, "OIA Image and Bit Group Information" for complete explanations of these positions.

Return Parameters

Return Code	Explanation
0	The target presentation space is unlocked.
1	Your program is not currently connected to the emulated host session.
2	An error was made in specifying string length. OIA data was not returned.
4	OIA data was returned. The target presentation space is busy.
5	OIA data was returned. The target presentation space is locked.
9	An internal system error was encountered. OIA data was not returned.

For Home3270, refer to Chapter 4, "AHLLAPI Functions."

Copy Presentation Space (5)

The **Copy Presentation Space** function copies the contents of the host-connected presentation space into a data area that you define in your EEHLLAPI application program.

The **Copy Presentation Space** function translates the characters in the host source presentation space into ASCII. Attribute bytes and other characters not represented in ASCII normally are translated into blanks. (If you do not want the attribute bytes translated into blanks, you can override this translation with the ATTRB option under the **Set Session Parameters** (9) function.)

Call Parameters

Data String: Preallocated target area the size of your host presentation space. This is 1920 for Entry-Level DOS mode.

Length: NA (length of the presentation space is implied)

PS Position: NA

Return Parameters

Return Code	Explanation
0	The host presentation space contents were copied to the application program. The target presentation space was active, and the keyboard was unlocked.
1	Your program is not currently connected to the emulated host session.
2	The receiving string was too small to contain PS data.
4	The host presentation space contents were copied. The connected host presentation space was waiting for host response.
5	The host presentation space was copied. The keyboard was locked.
9	A system error was encountered.

For Home3270, refer to Chapter 4, “AHLLAPI Functions.”

Notes on Using This Function

1. A program written in Interpretive BASIC cannot use this function, because copying the host presentation space exceeds the maximum allowed string size of 255 bytes. Compiled BASIC programs, however, can use this function.
2. If you want to copy only a portion of the host presentation space, use the **Copy Presentation Space to String** (8) function.

Copy Presentation Space to String (8)

The **Copy Presentation Space to String** function is used to copy all or part of the host-connected presentation space into a data area that you define in your EEHLLAPI application program.

The offset of the string into the host presentation space is based on a layout in which the upper-left corner (row 1/column 1) is Location 1 and the bottom-right corner is the maximum screen size for the host presentation space. (For the Entry-Level DOS mode, for example, this would be 1920.) The value of offset + Length cannot exceed the maximum screen size for the host presentation space.

The “PS Position” calling parameter is used to pass the beginning offset of the data string to the function. The maximum length of the target string is 255 bytes for Interpretive BASIC. The requested length must not exceed the length of your application program’s preallocated buffer size.

The **Copy Presentation Space to String** function translates the characters in the host source presentation space into ASCII. Attribute bytes and other characters not represented in ASCII normally are translated into blanks. If you do not want the attribute bytes translated into blanks, you can override this translation with the ATTRB option under the **Set Session Parameters** (9) function.

Call Parameters

Data string: Target string (maximum 255 bytes for Interpretive BASIC).

Length: Length of the target data string.

PS Position: The beginning of your string is represented by the byte position within the host presentation space.

The value of Position + Length cannot exceed the maximum size of the host presentation space. This value cannot exceed 1920 for the Entry-Level DOS mode.

Return Parameters

The **Copy Presentation Space to String** function returns a string containing the host presentation space contents and one of the following return codes:

Return Code	Explanation
0	The host presentation space contents were copied to the application program. The target presentation space was active, and the keyboard unlocked.
1	Your program is not currently connected to the emulated host session.
2	An error was made in specifying string length.
4	The host presentation space contents were copied. The host presentation space was waiting for host response.
5	The host presentation space was copied. The keyboard was locked.
7	The host presentation space position is not valid.
9	A system error was encountered.

For Home3270, refer to Chapter 4, “AHLLAPI Functions.”

Copy String to Field (33)

The **Copy String to Field** function transfers a string of characters into a specified field in the host-connected presentation space. This function can be used only in *a field-formatted host presentation space*.

The string to be transferred is specified with the Calling Data String parameter. The string ends when one of these four conditions is encountered:

When an EOT is encountered in the string (if EOT mode was selected using the **Set Session Parameters** (9) function). See "Set Session Parameters (9)" on page 48.

When the number specified in the length is reached if not in EOT mode.

When an end of field is encountered in the field.

When the end of the host presentation space is reached.

In other words, there is no wrapping.

Call Parameters

Data String: String containing the data to be transferred to a target field in the host presentation space.

Length: Length of the source data string or an EOT in the data string if in EOT mode.

PS Position: Position of the field that is the target of the copy.

Return Parameters

Return Code	Explanation
0	The **Copy String to Field** function was successful.
1	Your program is not currently connected to the emulated host session.
5	The target field was protected or inhibited, or incorrect data was sent to the target field (such as field attribute).
6	Copy was completed, but data is truncated.
7	The host presentation space position is not valid.
9	A system error was encountered.

For Home3270, refer to Chapter 4, "AHLLAPI Functions."

Copy String to Presentation Space (15)

The **Copy String to Presentation Space** function copies an ASCII data string directly into the host presentation space at the location specified by the "PS Position" calling parameter.

Call Parameters

Data String: String of ASCII data to be copied into the host presentation space.

Length: Length of data string or an EOT in data string if in EOT mode.

PS Position: Position in the host presentation space to begin the copy; between 1 and 1920 for the Entry-Level DOS mode.

Return Parameters

Return Code	Explanation
0	The **Copy String to Presentation Space** function was successful.
1	Your program is not currently connected to the emulated host session.
5	The target presentation space was protected or inhibited, or incorrect data was sent to the target field (such as a field attribute byte).
6	Copy was completed, but data was truncated.
7	The host presentation space position is not valid.
9	A system error was encountered.

Notes on Using This Function

1. If you want to place the string data at a specific cursor location, use the **Query Cursor Location** (7) function first to get the PS Position of the cursor. Then place this value in the "PS Position" calling parameter.
2. The string ends when an EOT is encountered in the string (if EOT mode was selected using the **Set Session Parameters** (9) function). See "Set Session Parameters (9)" on page 48.
3. Even though the **Send Key** (3) function seems to accomplish the same purpose as the **Copy String to Presentation Space** function, the latter is much faster in answering prompts and entering commands. The concept behind the **Send Key** function is to emulate a terminal operator typing in data from the keyboard. Thus, it is too slow for applications that require large amounts of data for each operation. The **Copy String to Presentation Space** function provides a much faster input path to the host.
4. The keyboard mnemonics (see the **Send Key** (3) function) cannot be sent using the **Copy String to Field** function.
5. The source data (the string to be copied) can be no larger than 1920 characters.

For Home3270, refer to Chapter 4, "AHLLAPI Functions."

Disconnect Presentation Space (2)

The **Disconnect Presentation Space** function drops the connection between your EEHLLAPI application program and the host presentation space.

Call Parameters

Data String: NA

Length: NA

PS Position: NA

Return Parameters

Return Code	Explanation
0	The **Disconnect** function was successful.
1	You are not currently connected to the host presentation space.
9	A system error was encountered.

Notes on Using This Function

1. Once the **Disconnect Presentation Space** function has been called, functions that interact with the host presentation space are no longer valid (for example, Send Key, Wait, Reserve, and Release).
2. Your program operator should disconnect from the host presentation space before exiting your EEHLLAPI application program. **Failure to do so can result in problems when using some PC utilities, particularly file transfer**.
3. The **Disconnect Presentation Space** function does not reset the session parameters to the defaults. Your program operator must call the **Reset System** (21) function to accomplish this. For example, a keyboard locked using the **Reserve** function will remain locked after the application has been disconnected.
4. Two parameters under Function 9 affect **Disconnect**:

Parameter	Explanation
CONPHYS	During the Disconnect, jump to the PC session where the EEHLLAPI application is running. (Do a physical disconnect.)
CONLOG	During the Disconnect, do not jump to the PC session. Stay at the current presentation space — could be at the HOST or PC. (Do a logical disconnect.)

For Home3270, refer to Chapter 4, “AHLLAPI Functions.”

Find Field Length (32)

The **Find Field Length** function returns the length of a target field in the connected presentation space. This function can be used to find either protected or unprotected fields, but only in *a field-formatted presentation space*.

This function returns the number of characters contained in the "Requested Field." This includes all characters from the beginning of the target field up to either the character preceding the next attribute byte or the end of the host presentation space.

Call Parameters

Data String: See table below.

Length: NA (2 is implied).

PS Position: Position within the host presentation space from which to start the Find.

The calling two-character data string can contain:

Code	Explanation
Blanks or "T "	This field
"P "	The previous field, either protected or unprotected
"N "	The next field, either protected or unprotected
"NP"	The next protected field
"NU"	The next unprotected field
"PP"	The previous protected field
"PU"	The previous unprotected field

Return Parameters

Length: = 0 means PC or unformatted host presentation space.

> 0 means length of requested field in host presentation space.

Return Code: The following return codes are valid:

Return Code	Explanation
0	The **Find Field Length** function was successful.
1	Your program has not issued a CONNECT to the emulated host session.
2	A parameter error was encountered.
7	The host presentation space position is not valid.
9	A system error was encountered.
24	No such field was found.

For Home3270, refer to Chapter 4, "AHLLAPI Functions."

Find Field Position (31)

The **Find Field Position** function returns the beginning position of a target field in the host-connected presentation space. This function can be used to find either protected or unprotected fields but only in *a field-formatted host presentation space*.

Call Parameters

Data String: See table below.

Length: NA (2 is implied).

PS Position: Position (within the field) relative to the origin of the host presentation space at which to start the Find.

The calling two-character data string can contain:

Code	Explanation
Blanks or "T "	This field
"P "	The previous field, either protected or unprotected
"N "	The next field, either protected or unprotected
"NP"	The next protected field
"NU"	The next unprotected field
"PP"	The previous protected field
"PU"	The previous unprotected field

Return Parameters

Length: = 0 means unformatted presentation space.

> 0 means relative position of requested field from the origin of the host presentation space. This position is defined to be the first position after the attribute byte.

Return Code: The following return codes are valid:

Return Code	Explanation
0	The **Find Field Position** function was successful.
1	Your program has not issued a CONNECT to the emulated host session.
2	A parameter error was encountered.
7	The host presentation space position is not valid.
9	A system error was encountered.
24	No such field was found.

For Home3270, refer to Chapter 4, "AHLLAPI Functions."

Pause (18)

The **Pause** function waits for a specified amount of time. It should be used in place of "timing loops" to wait for an event to occur. A **Pause** can be ended by a host event if a prior **Start Host Notification** (23) function had been called.

Call Parameters

Data String: NA

Length: Contains the pause duration in half-second increments.*

PS Position: NA

* A practical maximum value for **Pause** is 2400. You should not use **Pause** for these kinds of tasks:

Delay for very long durations (of several hours, for example)

With applications requiring a high-resolution timer. Since the time interval created by a **Pause** is approximate, such applications should use an alternate timing method.

Return Parameters

Return Code	Explanation
0	The wait duration has expired.
9	An internal system error was encountered. The time results are unpredictable.
26	A host session presentation space or OIA has been updated. Use the **Query Host Update** (24) function for more information.

Notes on Using This Function

One of the parameters set using Function 9 **Set Session Parameters** affects the length of the pause you get when you call this function.

Parameter	Definition
FPAUSE	Full-duration pause.
IPAUSE	Interruptible pause. the **Start Host Notification** (23) function and a host event will satisfy a Pause.

Once a pause has been satisfied by a host event, you *must* call the **Query Host Update** (24) function to obtain more data regarding the host PS/OIA update prior to the next **Pause**. If you use the IPAUSE option, **Pause** will continue to be satisfied with the pending event until the **Query Host Update** (24) function is completed.

Query Cursor Location (7)

The **Query Cursor Location** function indicates the position of the cursor in the host-connected presentation space by returning the cursor position.

Before you can call the **Query Cursor Location** function you must be connected to the host presentation space.

Call Parameters

Data String: NA

Length: NA

PS Position: NA

Return Parameters

Length: Presentation space position of the cursor.

Return Code: One from the table below:

Return Code	Explanation
0	The **Query Cursor Location** function was successful.
1	Your program is not currently connected to the emulated host session.
9	A system error was encountered.

Query Field Attribute (14)

The **Query Field Attribute** function returns the attribute byte of the field containing the host presentation space position in the host-connected presentation space. This information is returned as the "Returned Data Length" parameter. Note also these use characteristics:

The returned Data Length parameter is set to "0" if the screen is unformatted.

Attribute bytes are equal to or greater than hex C0.

Call Parameters

Data String: NA

Length: NA

PS Position: Presentation space position in the connected presentation space.

Return Parameters

Length: The Attribute value or "0" if the screen is unformatted.

Return Code: The following return codes are valid:

Return Code	Explanation
0	The **Query Field Attribute** function was successful.
1	Your program has not issued a CONNECT to the emulated host session.
7	The host presentation space position is not valid.

For Home3270, refer to Chapter 4, "AHLLAPI Functions."

Query Host Update (24)

The **Query Host Update** function lets the program operator determine if the host has updated the host PS/OIA since the last time this request was made. The target presentation space must be specified in the data string, even though you do not need to be connected to the host presentation space to check for updates.

The **Start Host Notification** function must be called before your program operator can use the **Query Host Update** function.

Call Parameters

Data String: One-character short name of the host presentation space.

Length: NA (1 is implied).

PS Position: NA

Return Parameters

Return Code	Explanation
0	No updates have been made since the last call.
1	An incorrect host presentation space was specified, one that was labeled with an incorrect name.
8	No prior **Start Host Notification** function was called for the host presentation space ID.
9	A system error was encountered.
23	The OIA and the host presentation space were updated. The Entry-Level DOS Mode Program cannot distinguish between OIA and presentation space updates.

Query Session Status (22)

The **Query Session Status** function provides session-specific information.

Call Parameters

Data String: Short name of the target presentation space plus 17 bytes for returned data.

Length: 18 bytes

PS Position: NA

Return Parameters

Data String: A data string of 18 bytes is returned. The bytes are defined below.

Byte	Definition
Position 1	Short name (PSID). This can contain: The one-letter short name of the host presentation space A blank or null indicating a function call against the host presentation space (if connected). If a "generic" request has been made, the actual PSID will be substituted in the string that is returned.
Positions 2–9	Reserved
Position 10	Session type (where C=CUT Host)
Position 11	For the Entry-Level DOS Mode Program, the returned value will always be 0.
Position 12	Number of rows in the host presentation space, expressed as a binary number.
Position 14	Number of columns in the host presentation space, expressed as a binary number.
Positions 16–17	Reserved
Position 18	Reserved

Return Code:

Return Code	Explanation
0	The **Query Session Status** function was successful.
1	The session requested was not valid.
2	A string length was not correct. This code will not be returned for all programming languages.
9	A system error was encountered.

Query Sessions (10)

The **Query Sessions** function, in the case of the Entry-Level DOS mode, returns a 12-byte data string describing the host session. The data string contains the short name, session type, and presentation space size of the host session.

Call Parameters

Data String: Preallocated string of 12 bytes.

Length: From 12 bytes.

PS Position: NA

Return Parameters

Data String: The returned data string contains the short name, host session type, and PS size of the host session.

The format of each 12-byte session descriptor is as follows:

Byte	Definition
Position 1	Short name of session. For Entry Level Emulation Program, this is E.
Positions 2–9	Long name of session. For Entry Level Emulation Program this is HOST1.
Position 10	Session type. For Entry Level Emulation Program, this is H.
Positions 11–12	Host presentation space size (this is a binary number and is not in display format).

Length: The number of configured sessions will always be 1 for Entry Level.

Return Code: The following return codes are valid:

Return Code	Explanation
0	The **Query Sessions** function was successful.
2	An improper string size was specified; the string is too small. (Note that here and in other functions the ability to verify actual string size is language dependent.)
9	A system error was encountered.

Notes on Using This Function

A maximum of 1 session (host) can exist in an Emulator environment.

Query System (20)

An EEHLLAPI application program can use the **Query System** function to determine the level of Entry Level support and other system-based values. This function returns a string with the appropriate system data. Most of this information is for use by a service coordinator when you call the IBM Support Center after receiving a Return Code 9 (a system error was encountered).

The bytes in this returned string are defined below under “Values Returned.”

Call Parameters

Data String: Preallocated string of 35 bytes.

Length: NA (Implied length is 35).

PS Position: NA

Return Parameters

Data String: A data string of 35 bytes is returned. The bytes are defined as follows:

Byte	Definition
Position 1	EEHLLAPI version number
Positions 2–3	EEHLLAPI level number
Positions 4–9	EEHLLAPI date (month, date, year–for service purposes only).
Position 10	LIM version number (only valid for LIMs provided on the Entry Emulator diskette; otherwise, LIM number=0).
Positions 11–12	LIM level number
Position 13	Hardware base, where P= PC or PC XT, A= Personal Computer AT*, and U=Unable to Determine.
Position 14	Control program type, where E=Entry Emulator.
Position 15	Control program level, where if an E is returned, then 1=1.XX.
Positions 16–19	Reserved
Positions 20–23	Extended Error Code 1. This is a printable ASCII string representing a hex word giving the EEHLLAPI component ID and system error number for that function (for service purposes only).
Positions 24–27	Extended Error Code 2. This is a printable ASCII string representing a fault symptom code for the last internal system error.
Positions 28–35	Reserved.

In general, extended error codes represent internal diagnostic information, such as the contents of the computer register that indicated the system error and contained the internal error code.

Return Code:

Return Code	Explanation
0	The **Query System** function was successful; data string has been returned.
2	Improper string size (string too small). Note that here and in other functions the ability to verify actual string size is language dependent.
9	A system error was encountered.

Notes on Using This Function

Your EEHLLAPI program should include a check of the return code for all function calls as a prerequisite for continuing your program.

If the return code is good, the program should continue.

If the return code is "9" (system error), your EEHLLAPI program should call a subroutine that requests the **Query System** (20) function. The subroutine could extract extended error code information to help your service coordinator determine the cause of the system error. When you contact your service coordinator, be prepared to give the system-error information generated by the **Query System** (20) function.

For Home3270, refer to Chapter 4, "AHLLAPI Functions."

Receive File (91)

The **Receive File** function is used to receive a file from the host session to the PC workstation session. It is used the same way as the “Receive” command is used in Entry Level. However, the **Receive File** function can be called by an EEHLLAPI application program.

Call Parameters

Data String: The same receive parameters as are usual for an Entry Level Emulation Control Program file transfer request.

Length: Length of the data string or EOT in the data string.

PS Position: Drive number

The drive number indicates the location of the RECEIVE.COM file, where:

Drive A = 1
Drive B = 2
.
.
.
Drive Z = 26

If RECEIVE.COM is located on a hard disk, the file must be in the current subdirectory.

Return Parameters

You can receive three kinds of return codes:

1. EEHLLAPI Return codes

   If a problem was encountered with the way you specified your data string or with the system, you would receive one of these return codes:

Return Code	Explanation
2	You have specified a data string length that is too long for the EEHLLAPI buffer. The file transfer was unsuccessful.
9	A system error was encountered.

2. File Transfer Message Codes

   The message number of the Entry Level Emulation Program file transfer messages can be returned. For example, if you are using the Entry Level Emulation Program, file transfer message "TRANS03 File transfer complete" would send a return code of 3. For a complete list of the file transfer messages, refer to Appendix A in *Personal Communications Version 4.1 Up and Running for DOS (Entry-Level Mode)*.

   An example of a message you might receive is listed below:

Return Code	Explanation
3	File transfer complete.

3. DOS Extended Error Codes

   DOS extended error codes are preceded by the number "3" (added by EEHLLAPI). To determine the DOS extended error code, subtract 300 and reference the *IBM DOS Technical Reference Guide*.

Notes on Using This Function

1. You must not be currently connected or have the session reserved before using the **Receive File** function. If you are linked to the host session already, you will receive File Transfer Message 7, “Cannot link to host: file transfer cancelled.”

   Therefore:

   If you are connected (with the **Connect Presentation Space** (1) function) to the host session, you must disconnect (using the **Disconnect Presentation Space** (2) function).

   If you have reserved (with the **Reserve** (11) function) the host session, you must release it (using the **Release** (12) function).

2. Special considerations for providing a path back to COMMAND.COM exist when you use this function.

   The parent application (EEHLLAPI) causes a subprocess program (like RECEIVE.COM) to be loaded, and the subprocess program inherits the parent’s environment. The EEHLLAPI environment segment contains the path back to the COMMAND.COM used when EEHLLAPI was loaded.

   While some subprocess applications are running, COMMAND.COM might be overlaid and might have to be reloaded. Therefore, the COMMAND.COM must be available on the same path as EEHLLAPI.

   It also means that RECEIVE.COM must be in the current directory when you issue a call to the **Receive File** (91) function. This is not necessarily the root directory or the directory in use when EEHLLAPI was loaded, all of which might be changed by the time the **Receive File** function is executed. RECEIVE.COM must be in the **current** directory.

3. Two sets of parameters under the **Set Session Parameters** (9) function are related to this function:

Parameter	Explanation
QUIET	Keeps SEND and RECEIVE messages from being displayed. EEHLLAPI will keep track of the message number and discard the message.
NOQUIET	Restores the display of messages.

Parameter	Explanation
TIMEOUT=N	A one-character indicator from the table below will tell EEHLLAPI how many 30-second cycles (how many messages with TRANS010) it should accept before issuing a CTRL+BREAK itself.

Character	Value (in minutes)	Character	Value (in minutes)
1	.5	8	4.
2	1.	9	4.5
3	1.5	J	5.
4	2.	K	5.5
5	2.5	L	6.
6	3.	M	6.5
7	3.5	N	7.

TIMEOUT=0	Timeout messages will be displayed every 30 seconds until the operator presses CTRL+BREAK (these messages would not be visible in the QUIET mode). This is standard for operator usage of SEND and RECEIVE.

Considerations for Using Functions 90 and 91

Functions 90 and 91 provide powerful additional abilities to high-level language applications, but they might require some effort to be usable in the language in which you choose to write your EEHLLAPI program.

This is because when DOS loads a program, it normally loads it so that a new program “owns” all of memory from the Program Segment Prefix (PSP) to the highest end of memory, including the memory occupied by COMMAND.COM (which contains the loader).

A well-behaved program uses the DOS SETBLOCK function call when it receives control from COMMAND.COM to shrink its allocated memory block down to the size it really needs. This action frees unneeded memory, which can then be used for loading subsequent programs (and the loader, if necessary).

You should also be aware that the MAX ALLOC field and MIN ALLOC field parameters in the EXE file header affect the memory size.

Functions 90 and 91 can only operate if there is enough free memory for the appropriate program to be loaded. If insufficient space is available, EEHLLAPI will return a 308 return code (“Insufficient memory”). In general, Interpretive BASIC and C do not occupy all of memory but might still produce a 308 return code if sufficient memory is not available for the subsequent program.

If left unchanged, Compiled BASIC, Pascal, and COBOL, when linked with their appropriate LIMs and loaded, will be given all of the available memory in the DOS session. Since there is no uniform way for EEHLLAPI to perform a SETBLOCK function for all languages and all situations, it does not do it. It is your responsibility to implement the DOS SETBLOCK function (DOS interrupt X'21' AH=X'4A') suitable to your high-level language to shrink the application module to a minimum size so that SEND.COM and RECEIVE.COM can be executed.

For more information, refer to the *IBM Personal Computer Disk Operating System Technical Reference*.

Release (12)

The **Release** function unlocks the host presentation space that was reserved using Function 11 **Reserve**.

Call Parameters

Data String: NA

Length: NA

PS Position: NA

Return Parameters

Return Code	Explanation
0	The **Release** function was successful.
1	Your EEHLLAPI program is not connected to a valid host presentation space.
9	A system error was encountered.

Notes on Using This Function

If you forget to **Release** a reserved host presentation space (reserved by using the **Reserve**(11) function) before ending your EEHLLAPI application, you will be locked out of that session until another application releases it or until you call the **Reset System** (21) function.

Reserve (11)

The **Reserve** function locks the host presentation space to prevent input from the terminal operator.

The reserved host presentation space remains locked until it is either unlocked by the **Release** (12) function or the **Reset System** (21) function.

Call Parameters

Data String: NA

Length: NA

PS Position: NA

Return Parameters

Return Code	Explanation
0	The **Reserve** function was successful.
1	Your EEHLLAPI program is not connected to a valid host presentation space.
9	A system error was encountered.

Notes on Using This Function

If your EEHLLAPI application program is sending a series of transactions to the host, you might need to prevent the user from gaining access to that session until your application processing is complete.

Reset System (21)

The **Reset System** function reinitializes the resident interface module EEHLLAPI. Event notification is stopped. The session parameter options are reset to their defaults. The reserved host session is released. The host presentation space is disconnected.

You can use **Reset System** during initialization or at program termination to reset the system to a known initial condition.

Call Parameters

Data String: NA

Length: NA

PS Position: NA

Return Parameters

Return Code	Explanation
0	The **Reset System** function was successful.
9	A system error was encountered.

Search Field (30)

The **Search Field** function examines a field within the connected host presentation space for the occurrence of a string. If the target string is found, this function returns the decimal position of the string numbered from the beginning of the host presentation space. (For example, the “row 1, column 1” position is numbered “1,” or the “row 5, column 1” position is numbered “321.”)

This function can be used to search for either protected or unprotected fields but only in *a field-formatted host presentation space*.

This function returns “0” if the string is not found.

Call Parameters

Data String: Target string for search.

Length: Length of the target data string or an EOT in the data string if in EOT mode.

PS Position: Position within the host presentation space where the search is to begin. Valid only if SRCHFROM parameter has been specified using the **Set Session Parameters** (9) function.

Return Parameters

Length: = 0 means that the string was not found.

> 0 means that the string was found at the host presentation space position.

Return Code:

Return Code	Explanation
0	The **Search Field** function was successful.
1	Your program has not issued a CONNECT to the emulated host session.
7	The host presentation space is not valid.
9	A system error was encountered.
24	The search string was not found, or the screen was unformatted.

Notes on Using This Function

Two sets of parameters under the **Set Session Parameters** (9) function relate directly to search operations:

Parameter	Explanation
SRCHALL	The **Search** function will scan the entire host presentation space.
SRCHFROM	The **Search** function will start from a specified beginning position.

Parameter	Explanation
SRCHFRWD	The **Search** function will be performed in an ascending direction.
SRCHBKWD	The **Search** function will be performed in a descending direction. A Search will be satisfied if the first character of the requested string starts within the bounds specified for the Search.

You can use the **Set Session Parameters** (9) function to determine whether your searches will search forward (SRCHFRWD) or search backward (SRCHBKWD). This is important because search will not wrap from the bottom of the screen to the top.

For Home3270, refer to Chapter 4, “AHLLAPI Functions.”

Search Presentation Space (6)

The **Search Presentation Space** function lets your EEHLLAPI program examine the host presentation space for the occurrence of a specified string.

Call Parameters

Data String: Target string of Search.

Length: Length of the target data string or an EOT in the data string if in EOT mode.

PS Position: Position within the host presentation space where the search is to begin. Valid only if SRCHFROM parameter has been specified using the **Set Session Parameters** (9) function.

Return Parameters

Length: = 0 means that the string was not found.

> 0 means that the string was found at the host presentation position.

Return Code:

Return Code	Explanation
0	The **Search Presentation Space** function was successful.
1	The host presentation space was not connected.
2	An error was made in specifying parameters (for example, the EOT character was specified under the **Set Session Parameters** function, but not found in the data string).
7	The host presentation space position is not valid.
9	A system error was encountered.
24	The search string was not found.

Notes on Using This Function

1. **Search Presentation Space** scans the host presentation space for the ***first*** occurrence of the specified string.

   If the string is not located, then the Returning Length is set to 0.

   If the string is found, then the Returning Length is set to the string's beginning location in the host presentation space. This location represents a position in the host presentation space based on the layout where the upper left corner ("row 1, column 1") is location 1 and the bottom right location is 1920 for Entry Level.

2. The **Search Presentation Space** function is useful in determining when the host presentation space in available. If your program operator is expecting a specific prompt or message before sending data, the **Search Presentation Space** function allows you to check for the prompt message before continuing. If the expected prompt has not yet been sent, your program can call the **Pause** (18) function or the **Query Host Update** (24) function and continue to call the **Search Presentation Space** function until a zero return code is received.

3. Two sets of parameters under the **Set Session Parameters** (9) function relate directly to Search functions:

Find Field Position

Parameter	Explanation
SRCHALL	The **Search** function will scan the entire host presentation space.
SRCHFROM	The **Search** function will start from a specified beginning position.

Parameter	Explanation
SRCHFRWD	The **Search** function will be performed in an ascending direction.
SRCHBKWD	The **Search** function will be performed in a descending direction. A Search will be satisfied if the first character of the requested string starts within the bounds specified for the Search.

The **Search Presentation Space** function normally checks the entire host presentation space. However, you can use Function 9 to specify SRCHFROM. In this mode, you specify a starting position for the search operation. Then the function looks for the designated string from that starting position through the end of the host presentation space. This option is useful if you are looking for a keyword that might have multiple occurrences in the host presentation space.

You can also use Function 9 to specify SRCHBKWD. Then the search operation locates the **last** occurrence of the string.

Send File (90)

The **Send File** function is used to send a file from the PC session where EEHLLAPI is running to a host session. It is used the same way the “Send” command is used in Entry Level. However, the **Send File** function might be called by an EEHLLAPI application program.

Call Parameters

Data String: The same send parameters as are usual for the Entry Emulator file transfer request.

Length: Length of the target data string or EOT in the data string if in EOT mode.

PS Position: Drive number.

The drive number indicates the location of the SEND.COM file where:

Drive A = 1
Drive B = 2
.
.
.
Drive Z = 26

If SEND.COM is located on a hard disk, the file must be in the current subdirectory.

Return Parameters

You can receive three kinds of return codes:

1. EEHLLAPI Return Codes

   If a problem was encountered with the way you specified your data string or with the system, you will receive one of these return codes:

Return Code	Explanation
2	You have specified a data string length that is too long for the EEHLLAPI buffer. File transfer was unsuccessful.
9	A system error was encountered.

2. File Transfer Message Codes

   The message number of the Entry Level file transfer messages can be returned. For example, if you are using Entry Level, file transfer message "TRANS03 File transfer complete" would send a return code of 3. For a complete list of the file transfer messages, refer to Appendix A in *Personal Communications Version 4.1 Up and Running for DOS (Entry-Level Mode)*.

   The most common messages you might receive are listed below:

Return Code	Explanation
3	File transfer complete.
4	File transfer complete with records segmented.

3. DOS Extended Error Codes

   The following DOS extended error codes are valid, preceded by the number "3" (added by EEHLLAPI).

Return Code	Explanation
301	Incorrect function number
302	File not found
305	Access denied
308	Insufficient memory
310	Incorrect environment
311	Incorrect format

Notes on Using This Function

1. You must not have issued a connect or reserve function call before using **Send File**. If you are linked to the host session already, you will receive File Transfer Message 7, “Cannot link to host: file transfer cancelled.”

   Therefore:

   If you are connected (with Function 1 **Connect Presentation Space**) to the host session, you must disconnect (using Function 2 **Disconnect Presentation Space**).

   If you have reserved (with Function 11 **Reserve**) a host session, you must release it (using Function 12 **Release**).

2. Special considerations for providing a path back to COMMAND.COM exist when you use this function.

   The parent application (EEHLLAPI) causes a subprocess program (like SEND.COM) to be loaded, and the subprocess program inherits the parent’s environment. The EEHLLAPI environment segment contains the path back to the COMMAND.COM used when EEHLLAPI was loaded.

   While some subprocesses applications are running, COMMAND.COM might be overlaid and might have to be reloaded. Therefore, the COMMAND.COM must be available on the same path as EEHLLAPI.

   It also means that SEND.COM must be in the current directory when you issue a call to Function 90 **Send File**. This is not necessarily the root directory or the directory in use when EEHLLAPI was loaded, both of which might be changed by the time the **Send File** function is executed. SEND.COM must be in the *current* directory.

3. Two sets of parameters under Function 9 **Set Session Parameters** are related to this function:

Parameter	Explanation
QUIET	Keeps SEND and RECEIVE messages from being displayed. EEHLLAPI will keep track of the message number and discard the message.
NOQUIET	Restores the display of messages.

Parameter	Explanation
TIMEOUT=N	A one-character indicator from the table below will tell EEHLLAPI how many 30-second cycles (how many messages with TRANS010) it should accept before issuing a CTRL+BREAK itself.
TIMEOUT=0	Timeout messages will be displayed every 30 seconds until the operator presses CTRL+BREAK (these messages would not be visible in the QUIET mode). This is standard for operator usage of SEND and RECEIVE.

Character	Value (in minutes)	Character	Value (in minutes)
1	.5	8	4.
2	1.	9	4.5
3	1.5	J	5.
4	2.	K	5.5
5	2.5	L	6.
6	3.	M	6.5
7	3.5	N	7.

For additional information, see “Considerations for Using Functions 90 and 91” on page 33.

Send Key (3)

The **Send Key** function sends a keystroke or a string of keystrokes to the host presentation space. Your EEHLLAPI application program must use The **Connect Presentation Space** (1) function before sending keystrokes.

You define the string of keystrokes to be sent with the calling data string parameter. The keystrokes appear to the target session as though they were entered by the terminal operator. You can also send all attention identifier (AID) keys such as Enter, PA1, and so on. All the fields that are protected for input or are numeric only must be treated accordingly.

Call Parameters

Data String: A string of keystrokes, maximum 255.

Length: Length of the target data string or an EOT in the data string if in EOT mode.

PS Position: NA

Return Parameters

Return Code	Explanation
0	The keystrokes were sent; status is normal.
1	Your program did not issue a connect to the emulated host session.
2	An incorrect parameter was passed to EEHLLAPI.
4	The host session was busy; all of the keystrokes could not be sent.
5	Input to the target session was inhibited or rejected; all of the keystrokes could not be sent.
9	A system error was encountered.

Note: When this function is issued with the attached control unit turned off, the return code is 9.

Notes on Using This Function

The following guidelines might help you use the **Send Key** function more efficiently:

Keystrokes cannot be sent to a session whose keyboard is locked; for example, when input in inhibited. You can check this with the **Wait** (4) function.

Keystroke input is no longer accepted after the first AID character is received. The remainder of the input data string might be rejected if the host is busy.

To send special Control keys, a compound character coding scheme is used. This coding scheme uses two ASCII characters to indicate one keystroke and compromises the @ sign followed by the keycode. For example, to type in the sequence “LOGON ABCDE” followed by the Enter key, you would code the string “LOGON ABCDE@E.” A complete list of these keycodes is represented in “Keyboard Mnemonics” on page 46.

This compound coding technique allows an ASCII string representation of all necessary keystroke codes without requiring the use of complex hexadecimal key codes. The string length passed to the application should count these compound keystrokes as being 2 bytes long.

You can specify an escape character other than "@" using the **Set Session Parameters** (9) function and specifying another character under "ESC=n."

The maximum length of a string that can be passed to the **Send Key** function in one request is 255 characters. However, your application program can make multiple calls to the **Send Key** function to transmit longer keystroke strings.

Users needing higher performance should use the **Copy String to Field** (33) function or the **Copy string to Presentation Space** (15) function rather than send keystrokes with Function 3.

Keyboard Mnemonics

This set of keyboard mnemonics is provided to allow you to use ASCII characters to represent the special function keys of the PC keyboard. A mnemonic abbreviation code makes these special keys easy to remember. An alphabetic key code has been used for the most common keys. For example, the Clear key is "C," the Tab key is "T," and so on. Note that the upper and lower case alphabetic characters are mnemonic abbreviations for different keys.

@B Backtab
@C Clear
@D Delete
@E Enter
@F Erase EOF
@I Insert
@L Cursor Left
@N New Line
@P Print
@R Reset
@T Tab
@U Cursor Up
@V Cursor Down
@Z Cursor Right

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

@A@C - Test
@A@D - Word Delete
@A@F - Erase Input
@A@H - System Request
@A@I - Alt+Insert
@A@J - Cursor Select
@A@P - Ident
@A@Q - Attention
@A@R - Device Cancel
@A@E - ExSel
@A@d - DOC Mode
@A@e - Wrap
@A@f - Change Format
@A@m - Cursor Position
@S@x - Dup
@S@y - Field Mark

AID Key Mnemonics

The AID key mnemonics set by the **Send Key** (3) function are listed below:

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

@A@C - Test	@C - Clear
@A@H - System Request	@E - Enter
@A@J - Cursor Select	
@A@Q - Attn	

You can specify an escape character other than "@" using an EOT delimiter defined using the **Set Session Parameters** (9) function and specifying another character under "ESC=n."

Note: The length of the data string is explicitly defined. You can also define the length implicitly using an end-of-transmission (EOT) delimiter using the **Set Session Parameters** (9) function.

For Home3270, refer to Chapter 4, "AHLLAPI Functions" on page 61.

Set Session Parameters (9)

The **Set Session Parameters** function lets you change certain default session options in EEHLLAPI, the resident interface module.

Call Parameters

Data String: String containing the session parameters.*

Length: Explicit length of the data string (EOT mode is not allowed).

PS Position: NA

* The Calling Data String can contain any of the parameters below. The parameters should be placed on the Calling Data String line, separated by commas or blanks. The sets of parameters are explained in terms of the functions they affect. The default is underlined.

The next three sets of parameters affect **Copy** functions. The parameter **EOT=n** also affects the **Send File** (90) function and the **Receive File** (91) function:

Parameter	Explanation
ATTRB	Pass back all codes that do not have an ASCII equivalent as their original values.
NOATTRB	Convert all unknown values to blanks.

Parameter	Explanation
STRLEN	An explicit length will be passed for all strings.
STREOT	String lengths are not explicitly coded. They are terminated with an EOT (End of Text) character.

Parameter	Explanation
EOT=n	Allows you to specify the EOT character for string terminators (in STREOT mode). Binary zero is the default. Do not leave a blank after the equal sign.

The next two sets of parameters affect the **Send Key** (3) function:

Parameter	Explanation
ESC=n	Specify the escape character for keystroke mnemonics (@ is the default). Do not leave a blank after the equal sign. Blank is not a valid character.

Parameter	Explanation
AUTORESET	The application will attempt to reset all inhibited conditions by prefixing all strings of keys sent using the **Send Key** (3) function with a reset.
NORESET	Do not AUTORESET.

The next two sets of parameters affect **Search** functions:

Parameter	Explanation
SRCHALL	The **Search** function will scan the entire host presentation space.
SRCHFROM	The **Search** function will start from a specified beginning position.

Parameter	Explanation
SRCHFRWD	The **Search** function will be performed in an ascending direction.

Parameter	Explanation
SRCHBKWD	The **Search** function will be performed in a descending direction. A search will be satisfied if the first character of the requested string starts within the bounds specified for the search.

The next set of parameters relates to using TRACE to debug your EEHLLAPI program. Refer to "Using Trace to Help You Debug Your Program" on page 77 for more information about this parameter.

Parameter	Explanation
TRON	Turns trace on.
TROFF	Turns trace off. The trace function might conflict with messages on the screen from languages or applications that manage their own displays. For more information about Trace messages, refer to Appendix A, "EEHLLAPI Messages."

The next set of parameters affects the **Wait** (4) function:

Parameter	Explanation
TWAIT	The **Wait** function will wait up to a minute before timing out on XCLOCK or XSYSTEM.
LWAIT	The **Wait** function will wait until XCLOCK/XSYSTEM clears. This option is not recommended since control does not return to your application until the host is available.
NWAIT	The **Wait** function checks status and returns immediately (no wait).

The next set of parameters affects **Disconnect** functions:

Parameter	Explanation
CONPHYS	During the Connect, jump to the requested presentation space (do a physical connect). During the Disconnect, jump to the PC session where the EEHLLAPI application is running (do a physical disconnect).
CONLOG	During the Connect, do not jump to the requested presentation space (do a logical connect). During the Disconnect, do not jump to the PC session. Stay at the current presentation space — could be at the HOST or PC (do a logical disconnect).

This set of parameters affects the **Pause** (18) function:

Parameter	Explanation
FPAUSE	If full-duration pause, it will pause for however long you specified in **Set Session Parameters**.
IPAUSE	Interruptible pause. The **Start Host Notification** (23) function and a host event will satisfy a Pause.

The next two sets of parameters affect the **Send** (90) function and the **Receive** (91) function:

Parameter	Explanation
QUIET	Keeps SEND and RECEIVE messages from being displayed. EEHLLAPI will keep track of the message number and discard the message.
NOQUIET	Restores the display of messages.

Parameter	Explanation
TIMEOUT=N	A one-character indicator from the table below will tell EEHLLAPI how many 30-second cycles (how many messages with TRANS010) it should accept before issuing a CTRL+BREAK itself.

Character	Value (in minutes)	Character	Value (in minutes)
1	.5	8	4.
2	1.	9	4.5
3	1.5	J	5.
4	2.	K	5.5
5	2.5	L	6.
6	3.	M	6.5
7	3.5	N	7.

TIMEOUT=0	Timeout messages will be displayed every 30 seconds until the operator presses CTRL+BREAK (these messages would not be visible in the QUIET mode). This is standard for operator usage of SEND and RECEIVE.

The next two sets of parameters affect the **Copy Presentation Space** (5) function, the **Copy Presentation Space to String** (8) function and the **Copy Field to String** (34) function: Extended Attribute Buffer (EAB) includes Extended Character Attributes and Extended Field Attributes.

Parameter	Explanation
NOEAB	Pass data only, no EABs.
EAB	Pass the presentation space data with extended attribute bytes. For each character that appears on the screen, 2 bytes of data are passed. Therefore, a buffer twice the size of the presentation space must be preallocated; for example 2 X 1920 = 3840 for a 24-row by 80-column presentation space.

Parameter	Explanation
NOXLATE	EABs are not translated.
XLATE	EABs are translated to the PC color graphics adapter (CGA) format.

Return Parameters

Return Code	Explanation
0	The session parameters have been set.
2	The length of the parameter list is not valid.
9	A system error was encountered.

For Home3270, refer to Chapter 4, “AHLLAPI Functions” on page 61.

Start Host Notification (23)

The **Start Host Notification** function begins the process by which your EEHLLAPI application program determines if the host PS/OIA have been updated. This version of EEHLLAPI does not distinguish between PS and OIA updates.

After using this function, your application program can use the **Query Host Update** (24) function to determine what specific host event has occurred.

Call Parameters

Data String: Preallocated string. See Note below.

Length: The length of the host event buffer (256 recommended).

PS Position: NA

Note: The Calling Data String contains these elements:

Position	Definition
Position 1	One of the following values: A specific host presentation space short name (PSID) A blank or null indicating a request against the host presentation space.
Position 2	The character B asking for notification of both host presentation space and OIA updates.
Positions 3–6	The 4-byte address of a preallocated buffer space to be used internally for enqueuing and dequeuing of host events. The suggested size of this space is 256 bytes, specified in the calling “Length” parameter.*

* Internally, this address is represented as a 2-byte offset followed by a 2-byte segment address of the buffer.

All supported languages can use the **Storage Manager** (17) function Subfunction 1 **Get Storage** to obtain a buffer block of storage. The **Get Storage** function allows all languages to get additional storage and to free smaller blocks of storage as needed.

The **Storage Manager** (17) function Subfunction 1 **Get Storage** returns a 4-byte address in the “Data String” parameter that can be concatenated with the first data string bytes for this function.

The following restrictions exist for individual languages:

Pascal or C: You choose either to use the **Storage Manager** function to provide the address of the buffer or to provide your own address pointer capability to point to a buffer structure in your data area.

Interpretive or Compiled BASIC: You must use the **Storage Manager** function since BASIC variables do not remain in a fixed location.

COBOL: You can use the **Storage Manager** function to provide the needed 4-byte address. Otherwise, your EEHLLAPI program must fill the space with 4 bytes of binary zeros (low value) so that the COBOL LIM will know that the buffer following immediately after the 4 bytes of binary zeros is a continuing part of that data structure. In this case, the COBOL LIM will provide the address itself for a common EEHLLAPI interface.

Note: If you allocate buffer storage with any language within your EEHLLAPI program (rather than using the **Storage Manager** (17) function), you *must* call the **Stop Host Notification** (25) function before you exit EEHLLAPI. (You could also use the **Reset System** (21) function to accomplish the same purpose). The **Failure to do so might cause subsequent programs to fail unpredictably with storage overlays.**

Return Parameters

Return Code	Explanation
0	The **Start Host Notification** function was successful.
1	An incorrect host presentation space was specified.
2	An error was made in designating parameters.
9	A system error was encountered.

Notes on Using This Function

To use this function, the **Storage Manager** function can be utilized. A **Get Storage** call should be made. See “Storage Manager (17)” on page 55.

Stop Host Notification (25)

The **Stop Host Notification** function disables the capability of the **Query Host Update** (24) function to determine if the host PS/OIA have been updated. This function also stops host events from affecting the **Pause** function.

Call Parameters

Data String: One-character short name of the target presentation space ID.

Length: NA (implied length of 1 byte).

PS Position: NA

Return Parameters

Return Code	Explanation
0	The **Stop Host Notification** function was successful.
1	An incorrect host presentation space was specified, one labeled with an invalid name.
8	No previous **Start Host Notification** was issued.
9	A system error was encountered.

Notes on Using This Function

After issuing **Stop Host Notification**, the **Storage Manager** subfunctions “Free Storage” or “Free All Storage” should be utilized if a previous “Get Storage” call was issued during **Start Host Notification**. See “Storage Manager (17)” on page 55.

Storage Manager (17)

The **Storage Manager** function is used in EEHLLAPI to allocate or deallocate queue storage for the **Start Host Notification** (23) function.

The **Storage Manager** function solves a storage allocation problem created by BASIC, but can be used with EEHLLAPI application programs written in other supported languages as well.

Users of BASIC can use the **Storage Manager** function to overcome the problem of dynamic storage pools used by BASIC.

The following restrictions exist for individual languages:

Pascal or C: You choose either to use the **Storage Manager** function to provide the address of the buffer or to provide your own address pointer capability to point to a buffer structure in your data area.

Interpretive or Compiled BASIC: You must use the **Storage Manager** function since BASIC variables do not remain in a fixed location.

COBOL: You can use the **Storage Manager** function to provide the needed 4-byte address. Otherwise, your EEHLLAPI program must fill the space with 4 bytes of binary zeros (low value) so that the COBOL LIM will know that the buffer following immediately after the 4 bytes of binary zeros is a continuing part of that data structure. In this case, the COBOL LIM will provide the address itself for a common EEHLLAPI interface.

Note: If you allocate buffer storage with any language within your EEHLLAPI program (rather than using the **Storage Manager** (17) function, you *must* call the **Stop Host Notification** (25) function before you exit EEHLLAPI. (You could also use the **Reset System** (21) function to accomplish the same purpose). **Failure to do so might cause subsequent programs to fail unpredictably with storage overlays.**

The preferred way to allocate storage for use by EEHLLAPI functions is to use the **Storage Manager** function. Three calls can be made to **Storage Manager**:

Get Storage

Free Storage

Free All Storage

Since each of these subfunctions has its own calling and returning parameters and generates its own set of return codes, each will be discussed as an individual subfunction. These subfunctions are identified to EEHLLAPI by passing the subfunction number in the calling “PS Position” parameter.

Get Storage

The **Get Storage** function allocates a piece of the storage block to be used by an application as a queue structure for the **Start Host Notification** function.

Call Parameters

Data String: NA (preallocated to 4 bytes).

Length: Size (in bytes) of the requested storage area.

PS Position: 01 (for Get Storage).

Return Parameters

The subfunction returns three pieces of information:

The Data String contains the storage address expressed as two binary words in offset segment order. This string should be used in Positions 3–6 in the Data String defined for the **Start Host Notification** function.

The Length parameter contains the storage block ID.

The Return Code, which is one of the following values:

Return Code	Explanation
0	The requested storage was allocated.
1	You have requested more storage than is available.
2	The LIM to which your EEHLLAPI application program is linked does not support this function.
9	A system error was encountered.
10	Storage Manager is not available.

Notes on Using This Function

For EEHLLAPI the **Storage Manager** function is designed to simulate the full functions provided on the 3270 PC HLLAPI.

Attention

Users should ONLY use this function to allocate storage for the **Start Host Notification** (23) function. If you attempt to use the allocated storage for other purposes, the results are unpredictable.

Free Storage

The **Free Storage** function frees the block of storage allocated by the **Get Storage** subfunction. This function should be used after a **Stop Host Notification** function request to free "queue" memory created by a **Get Storage** call on the **Start Host Notification** function.

Call Parameters

Data String: NA (preallocated to 4 bytes).

Length: ID of the storage block to be freed.

PS Position: 02 (for Free Storage).

Return Parameters

The subfunction returns these return codes:

Return Code	Explanation
0	The storage has been freed.
2	The storage block ID was not valid, or the LIM to which your EEHLLAPI application program is linked does not support this function.
9	A system error was encountered.
10	The **Storage Manager** function is not available.

Once a block of storage has been allocated, the block remains the size of the original allocation. A subsequent request for storage will waste the excess storage in the previously allocated block.

Free All Storage

The **Free All Storage** function frees all allocated storage blocks. The storage blocks are regrouped into a single storage pool that is the same size as the original block of memory allocated.

Call Parameters

Data String: NA (preallocated to 4 bytes)

Length: NA

PS Position: 04 (for Free All Storage)

Return Parameters

The size of the largest available block of storage is returned in the “Length” parameter.

One of the following return codes is also returned:

Return Code	Explanation
0	The **Free All Storage** function was successful.
2	The LIM to which your EEHLLAPI application program is linked does not support this function.
9	A system error was encountered.
10	The **Storage Manager** function is not available.

Wait (4)

The **Wait** function checks the status of the host presentation space. If the session is waiting on a host response (indicated by XCLOCK or XSYSTEM), the **Wait** function will cause EEHLLAPI to wait up to one minute to see if the condition clears.

Call Parameters

Data String: NA

Length: NA

PS Position: NA

Return Parameters

The following return codes are valid:

Return Code	Explanation
0	The keyboard is unlocked and ready for input.
1	Your application program is not connected to a valid session.
4	Timeout while still in XCLOCK or XSYSTEM.
5	The keyboard is unlocked.
9	A system error was encountered.

Notes on Using This Function

The **Wait** function is used to give host requests like those made by the **Send Key** (3) function the time required to be completed. Using the **Set Session Parameters** (9) function, you can make the request in three ways:

Parameter	Explanation
TWAIT	The **Wait** function will wait up to a minute before timing out on XCLOCK or XSYSTEM.
LWAIT	The **Wait** function will wait until XCLOCK/XSYSTEM clears. **Note:** This option is not recommended since control does not return to your application until the host is available.
NWAIT	The **Wait** function checks status and returns immediately (no wait).

You can use this function to see if the host OIA is inhibited.

The **Wait** function is satisfied by the host unlocking the keyboard. This might not mean the transaction has been completed, so you should use **Search** combined with **Wait** to look for expected keyword prompts.

Find Field Position

Chapter 4. AHLLAPI Functions

The Asynchronous High-Level Language Application Program Interface (AHLLAPI) is designed based on the CUT mode High-Level Language Application Program Interface (EEHLLAPI) and works with Home3270.

Functional Differences between AHLLAPI and EEHLLAPI

The AHLLAPI functions implemented that are different from EEHLLAPI are:

- Copy Field to String (34)
- Copy OIA (13)
- Copy Presentation Space (5)
- Copy Presentation Space to String (8)
- Copy String to Field (33)
- Copy String to Presentation Space (15)
- Disconnect Presentation Space (2)
- Find Field Position (31)
- Find Field Length (32)
- Query Field Attribute (14)
- Query System (20)
- Search Field (30)
- Send Key (3)
- Set Session Parameters (9)

Copy Field to String (34)

In AHLLAPI, copying ends when:

- The end of the target string is exceeded,
- The end of the host presentation space is reached, or
- The end of the field is reached.

Since the whole presentation space is treated as one field, conditions 2 and 3 would be the same. This function always copies from position 1 and ends at 1920. The value of the PS position parameter is ignored.

Copy OIA (13)

Positions 2–81 of the returned data string contain the OIA image. This is in the ASCII format passed down by the attached protocol converter. The exact image of the OIA depends upon the particular protocol converter used.

© Copyright IBM Corp. 1996

Copy Presentation Space (5)

No attribute bytes are stored in the host presentation space, so attribute bytes cannot be copied using the ATTRB option of the **Set Session Parameters** (9) function.

Copy Presentation Space to String (8)

No attribute bytes are stored in the host presentation space, so attribute bytes cannot be copied using the ATTRB option of the **Set Session Parameters** (9) function.

Copy String to Field (33)

For Home3270, the whole presentation space is considered to be one field starting from position 1 and ending at 1920. This function always starts the copy from position 1 of the host presentation space and ignores the value of the PS position parameter being passed.

The string ends when:

an EOT is encountered in the string, if EOT mode was selected using the **Set Session Parameters** (9) function.

the number specified in the length is reached if not in EOT mode.

the end of the host presentation space (which is also the end of the file) is reached.

Copy String to Presentation Space (15)

The function implemented by AHLLAPI is different from that implemented by EEHLLAPI. EEHLLAPI puts the input ASCII string to the host presentation space without intervention to the attached control unit. AHLLAPI gets the function by performing another pass in the same manner as the **Send Key** (3) function.

To do the copy function, Home3270 transmits the characters to the protocol converter, which then echoes them back to the presentation space. The cursor is reset to the original position when the copy is completed. Home3270 never waits for the cursor to return to its original position before issuing a return code.

If successive copies are to be done, the application should wait until the cursor is returned to the original position before issuing the next copy function call. The **Query Cursor Location** (7) function can be used to check if the cursor is returned to the original position.

Disconnect Presentation Space (2)

This function drops the connection between the AHLLAPI application program and the host presentation space, and unlocks the keyboard if it was locked using the Reserve function.

Find Field Position (31)

This function always returns zero in the Length Parameter since the host presentation space in Home3270 is unformatted.

Find Field Length (32)

This function always returns zero in the Length Parameter since the host presentation space in Home3270 is unformatted.

Query Field Attribute (14)

This function always returns zero in the length parameter since the host presentation space in Home3270 is unformatted.

Query System (20)

Home3270 responds as follows to the **Query System** command:

Position	Description	Value
1	AHLLAPI Version Number	1 (31H)
2, 3	AHLLAPI Level Number	00 (3030H)
4 to 9	AHLLAPI Date (month, date, year)	MMDDYY
10	LIM Version Number	3 (33H)
11, 12	LIM Level Number	00 (3030H)
13	Hardware Base	Z (5AH)
14	Control Program Type	A (41H)
15, 16	Control Program Sequence Number	00 (3030H)
17, 18	Control Program Level	10 (3130H)
19	Session Short Name	E (45H)
20 to 23	Extended Error Code 1	NA
24 to 27	Extended Error Code 2	NA
28, 29	System Model and Submodel Numbers	by BIOS INT 15H AH=C0H
30, 31	National Language Support	Country code
32	Monitor Type	A = PS/2 Mono not Model 30 B = PS/2 Model 30 Mono C = CGA E = EGA G = PS/2 Model 30 Color M = Monochrome U = Unknown V = VGA
33 to 35	Reserved	

Search Field (30)

For Home3270, the whole presentation space is considered to be one file starting from position 1 and ending at position 1920. This affects the way a search is performed under different search parameter combinations set with the **Set Session Parameters** (9) function :

SRCHALL and SRCHFRWD

Searches from position 1 to 1920 (ignores the PS Position parameter) and finds the first occurrence.

SRCFROM and SRCHFRWD

Searches from position 1 (which is the beginning of the field) to 1920 (end of field) and finds the first occurrence.

SRCHALL and SRCHBKWD

Searches from position 1 to 1920 (ignores the PS Position parameter) and finds the last occurrence.

SRCFROM and SRCHBKWD

Searches from position 1 (which is the beginning of the field) to 1920 (end of field) and finds the last occurrence.

Send Key (3)

AHLLAPI does not support the following key mnemonics:

Change Format

Cursor Position

DOC Mode

ExSel

Word Delete

Wrap

Since Home3270 does not recognize fields, it cannot recognize protected and unprotected areas. The Send Key function should only be used within an unprotected field. If an application performs the function to a protected area, the return code will indicate a successful send but the protocol converter will lock the keyboard, the send keys will be truncated, and the cursor position will be unpredictable.

Home3270 treats any characters it receives from AHLLAPI as if they came from the keyboard, and does any national language translations required on the basis of the configuration before the characters are sent to the host or displayed on the screen.

Set Session Parameters (9)

For parameters ATTRB/NOATTRB, see “Copy Presentation Space (5)” on page 62 and “Copy Presentation Space to String (8)” on page 62.

For parameters SRCHALL, SRCHFROM, and SRCHBKWF, see “Search Field (30)” on page 64.

For parameters AUTORESET/NORESET, see “Send Key (3)” on page 64. The default value is NORESET.

Additional AHLLAPI Functions

Additional AHLLAPI functions not supported by EEHLLAPI are:

Drop Session DTR (82)

Query Emulator Host Connection (80)

Raise Session DTR (81)

Drop Session DTR (82)

This function enables an AHLLAPI application to have Home3270 disconnect the active host session (Home3270 will drop DTR).

Parameters Required When Called:

- Request Number = 82
- Data String = Short Name of Session ID (E)
- Length = (Implied length is 1)
- PS Position = NA

Values Returned by AHLLAPI:

- Data String = Short name of Session ID (E)
- Return Code (Decimal)
  - 0: DTR was successfully dropped
  - 1: Incorrect session ID
  - 9: System error

Query Emulator Host Connection (80)

This function enables an AHLLAPI application to determine if Home3270 has an active host session. If a host connection does not exist, this command also informs the AHLLAPI application whether automatic dials are defined.

Notes:

1. Home3270 requires both DSR** and CD to be present before it considers the host session active.
2. If Home3270 is configured so ROLM** DCM attachment is Yes, then only DSR must be present before it considers the host session active.

Parameters Required When Called:

- Request Number = 80
- Data String = Short Name of Session ID (E)
- Length = (Implied length is 1)
- PS Position = NA

Values Returned by AHLLAPI:

- Data String = Short name of Session ID (E)
- Return Code (Decimal)
  - 0: Session is connected
  - 1: Incorrect session ID
  - 9: System error
  - 32: Session is not connected (automatic dials defined)
  - 33: Session is not connected (no automatic dials defined)

Raise Session DTR (81)

This function enables an AHLLAPI application to have Home3270 connect to the host session (Home3270 will raise DTR).

Parameters Required When Called:

- Request Number = 81
- Data String = Short Name of Session ID (E)
- Length = (Implied length is 1)
- PS Position = NA

Values Returned by AHLLAPI:

- Data String = Short name of Session ID (E)
- Return Code (Decimal)
  - 0: DTR is successfully raised
  - 1: Incorrect session ID
  - 9: System error

Find Field Position

Chapter 5. Compiling and Running Your EEHLLAPI Application Program

You learned in Chapter 1 that when you have finished writing your EEHLLAPI application program, you can make the corresponding Language Interface Module (LIM) a permanent part of your program. This chapter teaches you:

How to link your EEHLLAPI application program with the appropriate LIM

How to run your application program

How to use Trace to help you debug your program

Linking Your EEHLLAPI Application Program with the Appropriate LIM

When you are ready to run your EEHLLAPI application program, you can make the appropriate LIM a permanent part of your program in this manner:

If you are using a compiled language (such as COBOL or Compiled BASIC), you call the appropriate language interface as an external subroutine. When you link-edit your program (using the DOS **link** command), the appropriate LIM will be merged with your EEHLLAPI program.

If you are using the Interpretive BASIC language, you will have to use another technique for calling the BASIC LIM. (Interpretive BASIC has no provision for using the DOS **link** command to include an external subroutine.) Instead, you must access a special Interpretive BASIC LIM that is available whenever EEHLLAPI is loaded.

Each language that you can use to call EEHLLAPI functions has its own programming conventions and requirements. These requirements are explained below for the languages supported directly by EEHLLAPI (those for which LIMs are provided). For information about writing a LIM to support another programming language, refer to Appendix B, “Writing Your Own Language Interface Module.”

Information about each directly supported high-level language follows.

Using BASIC

The EEHLLAPI function **Copy Presentation Space** (5) is not available to application programs written in Interpretive BASIC. Users of BASIC have a choice of two programming environments: Interpretive BASIC and Compiled BASIC. While these environments are similar, differences exist in subroutine linkage and string handling.

Interpretive BASIC

If you use Interpretive BASIC, you will need special initialization programming to access the Interpretive BASIC language interface that is in the EEHLLAPI module. The special code locates the interface from a fixed pointer location and uses this address to set the SEG and OFFSET values for the subroutine call.

If you do not understand the routine involved, do not worry; just copy the following program statements into your program's initialization routine:

© Copyright IBM Corp. 1996

```
1   DEF SEG =  : APILOC=PEEK(&H1FE) + PEEK(&H1FF)

11  IF APILOC =   THEN PRINT "EEHLLAPI IS NOT AVAILABLE": END
12  DATA &H 4B8, &HB3 1, &HCDBB, &HCA7F, &H   2
13  DIM APICALL% (4)
14  FPR I=  to 4: READ APICALL%(I): NEXT I
15  BLIMSET = VARPTR (APICALL%( ))
16  DEF SEG: CALL BLIMSET (APICALL%(1))
17  BLIM=APICALL%(1): DEF SEG = APICALL%( )

18  IF BLIM = &HB3 1 THEN PRINT "EEHLLAPI NOT AVAILABLE": END
```

Note: While the initialization code used by Interpretive BASIC is not required by Compiled BASIC, it is compatible with it, allowing you to develop a program with Interpretive BASIC and then to compile it to improve performance.

To call the interface from BASIC, you use a fixed format call with four parameters. These parameters must occur in a fixed sequence and all four parameters must be in the call list, even if they are not used. If a parameter is not needed for a given call, you do not need to value it.

The call format is:

```
1   CALL BLIM (FUNC%,SDATA$,DLEN%,RETC%)
```

Where:

FUN% is the function code
SDATA$ is a string of data
DLEN% is the data length
RETC% is the return code.

Note these restrictions in defining these parameters;

The parameters can have any names, but they **must** be of the proper type. Refer to the definitions of the calling parameters under “Call Parameters” on page 7 and of the returned parameters under “Return Parameters” on page 8.

The function code, data length, and return code **must** be 2-byte integer variables.

The data string **must** be a string variable.

Users of BASIC need to carefully review the restrictions for string processing.

Since the maximum string length for Interpretive BASIC is 255 bytes, the **Copy Presentation Space** (5) function is not available, but the **Copy Presentation Space to String** (8) function can be used to extract segments of the host presentation space.

String variables are managed dynamically in BASIC's data segment. This means that all strings **must** be preallocated before they can receive data from the interface.

While this might sound complicated, all it means is that the subroutine package **cannot** change the size of a string, so you must preformat your string to the required size.

Let's look at an example:

```
1      SDATA$=SPACE$(8 ) 'PREALLOCATE TO 8  BYTES
1 1    FUNC%= 8      'COPYSTRING  FUNCTION
1 2     DLEN%=8      'COPY 8  BYTES
1 3     RETC%=1      'LOCATIONS 1-8  FOR COPY
1 35    REM NOTE USE OF RETC% TO PASS OFFSET FOR COPYSTRING
1 4    CALL  BLIM(FUNC%,SDATA$,DLEN%,RETC%)
```

If you forget to preallocate your strings, the BASIC LIM attempts to detect this condition and will return a parameter specification error.

An issue related to string preallocation for Interpretive BASIC is string performance. If you are writing complex or long-running applications, you should understand the implications of Interpretive BASIC's string usage so that you can take steps to avoid degraded performance.

Interpretive BASIC manages strings dynamically. This means that as you assign and reassign data to a string, Interpretive BASIC reacquires space in its dynamic storage pool.

While this is not a problem for many programs, if you write a complex application that uses strings frequently, Interpretive BASIC might continue to append strings. This can result in lengthy pauses in your program's execution while BASIC compresses strings.

Compiled BASIC

Users of Compiled BASIC can call the Compiled BASIC Language Interface Module (BLIM) as an external call. To do this, use the **link** module provided with Compiled BASIC to link the Compiled BASIC LIM (named HLLCBAS.OBJ and located on your Entry Level diskette) with your EEHLLAPI application program. To do this, follow these general instructions:

1. Compile your EEHLLAPI program as instructed in the *IBM Compiled BASIC* manual.
2. Place your IBM BASIC Compiler "BASIC" diskette containing the file LINK.EXE in your default diskette drive.
3. Have your compiled EEHLLAPI program and the file HLLCBAS.OBJ from your Entry Level diskette available. You will need to specify the drive locations of these programs for the link program and add path information to the basic link command provided below.
4. Type LINK YOURPROG+HLLCBAS; after the DOS prompt.

This will produce an executable module named YOURPROG.EXE. After loading EEHLLAPI, you can execute the BASIC program in the usual manner.

For more details about linking programs under Compiled BASIC, refer to *IBM Personal Computer Language Series: BASIC Compiler*.

For programming samples in BASIC, refer to the individual functions and the BASIC program sampler.

Using COBOL

If your are writing your EEHLLAPI program in COBOL, be aware that the COBOL program should declare the parameter variables in the Data Division.

Here is an example of the parameter declaration:

```
77  FUNCODE          PIC  99          COMP- .
77  RETCODE          PIC  99          COMP- .
77  STRLEN           PIC  99          COMP- .
 1  DATA-STRING      PIC  X(192 )     VALUE  SPACES.
```

You have the option of changing the data names to fit your own programming conventions. The string variable should always be large enough to receive the largest amount of host data you will request.

The call to the COBOL LIM would be coded:

```
CALL 'HLLCOB' USING FUNCODE DATA-STRING STRLEN RETCODE.
```

Once you have compiled your EEHLLAPI program, you will need to use the DOS link command to link it with the COBOL LIM (named HLLCOB.OBJ on your Entry Level diskette).

To do this, follow these general instructions:

1. Compile your EEHLLAPI program as instructed in the *IBM Personal Computer Language Series: COBOL* manual.
2. Place your DOS “Supplemental Programs” diskette in your default diskette drive.
3. Have your compiled EEHLLAPI program and the file HLLCOB.OBJ from you Entry Level diskette available. You will need to specify the drive locations of these programs for the link program and add path information to the basic link command provided below.
4. Type `LINK YOURPROG+HLLCOB;` after the DOS prompt.

This will produce an executable module called YOURPROG.EXE. After loading EEHLLAPI you can execute this COBOL program in the normal manner.

For more details about linking programs under DOS, refer to the *IBM Personal Computer Language Series: Disk Operating System* manual. For more details about using COBOL, refer to *IBM Personal Computer Language Series: COBOL*.

Using Pascal

If you are writing your EEHLLAPI program in Pascal, you should note that Pascal needs to declare the required variables and to link to the Pascal language interface module. Following is an example of the necessary definitions:

```
VAR
 FUNC,                          ( Function number                    )
 LEN,                                ( Length of the data string        )
 RETC :INTEGER;                 ( EEHLLAPI return code               )
 CONN_PS_STR: STRING(1);
                                     ( Connect Presentation Space string )

PROCEDURE HLLPAS (VAR LIM_FUNC: INTEGER;
                  VAR  LIM_STR:    STRING;
                  VAR  LIM_LEN,
                       lIM_RETC: INTEGER); EXTERN;
```

The call to the LIM would then be coded:

```
HLLPAS (FUNC,CONN_PS _STR,LEN,RETC);
```

Several EEHLLAPI functions separate the data string, either on call or return, into a series of fields. Although IBM Pascal 2.0 provides no explicit substring functions for variables of type STRING, there are ways to work with the fields within the data string.

One suggested method is to use record overlays. By setting the address of a record equal to the address of the data string, you can directly access the different type fields within the data string. For example, a call to the **Query System** (20) function can be done as shown on the following page.

Find Field Position

```
PROGRAM QSYS(INPUT,OUTPUT);

TYPE
                                        ( Query System overlay type        )
                                        ( [xx] necessary for proper        )
  QSYSOVLTYP = RECORD                     (    byte alignment                  )
   HLLVERS [  ]: STRING(1);       ( EEHLLAPI version number          )
   HLLLVL    [ 1]: STRING(2);      ( EEHLLAPI level number            )
   HLLDATE [ 3]: STRING(6);       ( EEHLLAPI date                    )
   LIMVERS [ 9]: STRING(1);       ( LIM version number               )
  LIMUL     [1 ]:  STRING(2);      (  LIM  level  number              )
  HARDW     [12]:  STRING(1);      (  Hardware  base                  )
   CPTYPE    [13]: STRING(1);      ( Control Program type             )
  CPLVL     [14]:  STRING(1);      (  Control  Program  level         )
  RES1       [15]:  STRING(1);      (  Reserved                        )
  RES2       [16]:  STRING(2);      (  Reserved                        )
   PCSHRTN [18]: STRING(1);       ( PC short session name            )
   COMPRC    [19]: STRING(4);      ( Component error return code      )
  SYSRC     [23]:  STRING(4);      (  System  error  return  code     )
  RES3       [27]:  STRING(9);      (  Reserved                        )
  END;

VAR
 FUNC,                                  ( Function number                  )
 LEN,                                   ( Length of the data string        )
 RETC :INTEGER;                  ( EEHLLAPI integer                  )
 Q_SYS_OVL: QS YSOVLTYP;
                                        ( Query System Overlay             )
 Q_SYS_STR: STRING(35);
                                        ( Query System data string         )
 Q_SYS_PTR: ADS OF Q_SYS_OVL;
                                        ( pointer to the Q System overlay )


PROCEDURE HLLPAS (VAR LIM_FUNC: INTEGER;
                  VAR  LIM_STR:    STRING;
                  VAR  LIM_LEN,
                       lIM_RETC: INTEGER); EXTERN;

BEGIN
  ( ------------------------------------------------------- )
 (            Query  System   (Function  2 )
  ( ------------------------------------------------------- )

  FUNC := 2 ;
 LEN   :=  35;                                  (  Data  String  Length      )
  RETC :=  ;

 HLLPAS  (FUNC,Q_SYS_STR,LEN,RETC);

  Q_SYS_PTR := ADS OF Q_SYS_STR;
                                                  ( Set overlay on string  )
  WITH Q_SYS-PTR DO
   BEGIN
     WRITELN('EEHLLAPI version number:            ':23, HLLVERS ;
     WRITELN('EEHLLAPI level number:              ':23, HLLLVL    ;
     WRITELN('EEHLLAPI date:                       ':23, HLLDATE ;
     WRITELN('LIM version number:                 ':23, LIMVERS ;
```

```
    WRITELN('LIM level number:                ':23, LIMLVL    ;
   WRITELN('Hardware  base:                   ':23,  HARDW      ;
    WRITELN('Control Program type:            ':23, CPTYPE    ;
    WRITELN('Control Program level:           ':23, CPLVL      ;
    WRITELN('Base PC short name               ':23, PCSHRTN);
    WRITELN('Component return code:           ':23, COMPRC );
    WRITELN('System return code:              ':23, SYSRC     );
  END;

END.
```

Once you have compiled your EEHLLAPI program, you will need to use the DOS link command to link it with the Pascal LIM (named HLLPAS.OBJ on your Entry Level diskette).

To do this, follow these general instructions:

1. Compile your EEHLLAPI program as instructed in the *IBM Personal Computer Language Series: Pascal* manual.
2. Place your DOS “Supplemental Programs” diskette in your default diskette drive.
3. Have your compiled EEHLLAPI program and the file HLLPAS.OBJ from your Entry Level diskette available. You will need to specify the drive locations of these programs for the link program and add path information to the basic link command provided below.
4. Type LINK YOURPROG+HLLPAS; after the DOS prompt:

This will produce an executable module called YOURPROG.EXE. After loading EEHLLAPI you can execute this Pascal program.

For more details about linking programs under DOS, refer to the *IBM Personal Computer Language Series: Disk Operating System* manual. For more details about using Pascal, refer to *IBM Personal Computer Language Series: Pascal*.

Using IBM C

If you are writing your EEHLLAPI program in IBM C, you should note that IBM C needs to declare the required variables and to link to the IBM C language interface module. Following is an example of the necessary definitions:

```
INT API_FUNC, API _LEN,
      API_RETC
CHAR API-STRING [255]
```

The call to the language interface would then be coded:

```
HLLC(&API_FUNC, API _STR, &API_LEN, &API_RETC);
```

Once you have compiled your EEHLLAPI program, you will need to use the IBM C CLINK command to link it with one of three IMB C LIMS, depending on the memory model option specified during compilation:

HLLC_S.OBJ (small memory model)

HLLC_M.OBJ (medium memory model)

HLLC_L.OBJ (large memory model)

To do this, follow these general instructions.

1. Compile your EEHLLAPI program as instructed in the *IBM Personal Computer Language Series: C Compiler, Link and Run* manual.
2. Have your compiled EEHLLAPI program and the file "HLLC_X.OBJ" (where "X" is either S, M, or L, depending on memory model option selected) from your Entry Level diskette available. You will need to specify the drive locations of these programs for the link program and add path information to the basic link command provided below.
3. Type `CLINK YOURPROG+HLLC_X;` after the DOS prompt.

This will produce an executable module called YOURPROG.EXE. After loading EEHLLAPI, you can execute this IBM C program.

For more details about compiling and linking programs under DOS, refer to the *IBM Personal Computer Language Series: C Compiler Compile, Link and Run* manual.

Using 8088 Assembler

If you program in the Assembler language, you do not need a LIM. You can use software interrupts instead. If you are using an EEHLLAPI program written under this version of EEHLLAPI, you can invoke EEHLLAPI.EXE directly by issuing an Interrupt hex 7F with AH=01, AL=04, and BX=00.

At the time of the interrupt, DS:SI should point to a Parameter Control Block specifying your parameters. For more details on how to construct a parameter control block for Assembler language, see Appendix B, "Writing Your Own Language Interface Module."

Running Your EEHLLAPI Application Program

When you are ready to run your EEHLLAPI program with an application, your EEHLLAPI program should have the following things available:

1. Have your Entry Level and EEHLLAPI loaded into storage.
2. Be sure that any DOS files you plan to use are available to your EEHLLAPI program as specified by the individual functions. This includes the Entry Level utilities, such as:

   SEND.COM

   RECEIVE.COM

Using Trace to Help You Debug Your Program

If you need help tracing program events when you are debugging your EEHLLAPI application program, you might want to turn on Trace using the **Set Session Parameters** (9) function. The TRON and TROFF parameters are explained below.

Parameter	Explanation
TRON	Turns trace on.
TROFF	Turns trace off.

By turning Trace on, EEHLLAPI will indicate:

When a function begins execution
When the function completes
What code was returned

The Trace function might conflict with messages on the screen from languages or applications that manage their own displays. For more information about Trace messages, refer to Appendix A, “EEHLLAPI Messages.”

Find Field Position

Appendix A. EEHLLAPI Messages

Messages from the Personal Computer High-Level Language Application Program Interface can be identified by the prefix “EHL” and a message code followed by the message text. The message codes have the following format:

EHL*nnn Message Text*

Where *nnn* is a message ID.

The messages are listed below in numeric order:

EHL001 EEHLLAPI is loaded

Explanation: The program is now loaded. This message is followed by EHL002.

EHL002 EEHLLAPI is ready for use

Explanation: The program is successfully loaded and available for your use. This message is displayed after EHL001 or EHL003.

EHL003 EEHLLAPI is already loaded

Explanation: You entered the command **eehllapi** or **ahllapi** when the EEHLLAPI program was already loaded. This message is followed by EHL002.

Since you have already loaded EEHLLAPI, the program is available for use.

EHL004 Unable to load EEHLLAPI

Explanation: After you entered the command **eehllapi** or **ahllapi**, an error occurred while trying to load the program. The message that was displayed before this one indicates what went wrong. The program is not available for use.

EHL005 Incorrect level of the Entry Level Emulation Program.

Explanation: You have loaded an incorrect level of the Entry Level base program. This message is followed by EHL004. The program is not available for use.

EHL008 System error – 88nn

Explanation: An error with the Entry Level base program was encountered while trying to load EEHLLAPI. This message is followed by EHL004. The program is not available for use.

“88nn” is a four-digit return code. The first two digits are always “88” to indicate EEHLLAPI. Record this information and have it available when you talk with your system coordinator.

© Copyright IBM Corp. 1996

EHL010 EEHLLAPI trace. Request nn

Explanation: You specified TRON (Trace On) as a session parameter. This message indicates which request you have invoked. “nn” is the request number in hexadecimal notation. The trace messages might conflict (overlap) with your application messages if you are using a language like BASIC that manages its own display.

EHL012 EEHLLAPI trace. Return code: nnnn

Explanation: You specified TRON (Trace On) as a session parameter. This message indicates ending status for each request. These return codes are dependent upon the EEHLLAPI request that was executed. Note that this might overlap with messages from programs that track their own screen management (that is, BASIC). The value is displayed in hexadecimal notation.

Appendix B. Writing Your Own Language Interface Module

The Personal Computer High-Level Language Application Program Interface is designed to allow you to develop support for languages and features that might be unique to your environment. By using a language interface module (LIM) as a bridge between the application program and the primary interface program, you can:

Write a LIM for any language you might want to support.

Use an existing LIM as a gateway to other external functions.

Change the format of the function calls or parameter lists to better suit your environment.

The most basic step in getting ready to write your own LIM is to understand how the parts of EEHLLAPI work together. Refer to Figure 1 on page 82 as you read this information.

Let's look at a typical example of how your EEHLLAPI application program uses a EEHLLAPI function.

1. Your EEHLLAPI program calls the appropriate LIM.
2. The LIM takes your program's parameters and builds a **parameter control block**.
3. The LIM passes this control block and your EEHLLAPI request to the resident interface module (EEHLLAPI.EXE) by means of software interrupt hex 7F.
4. EEHLLAPI interprets the information and passes the request on to the Entry Level base program.
5. The Entry Level base program performs (or attempts) the task.
6. After the task is completed (or attempted), EEHLLAPI places a return code in the parameter control block and returns control to the calling LIM.
7. The LIM then returns control to your EEHLLAPI application program.

The path for the CUT connection looks something like this:

© Copyright IBM Corp. 1996

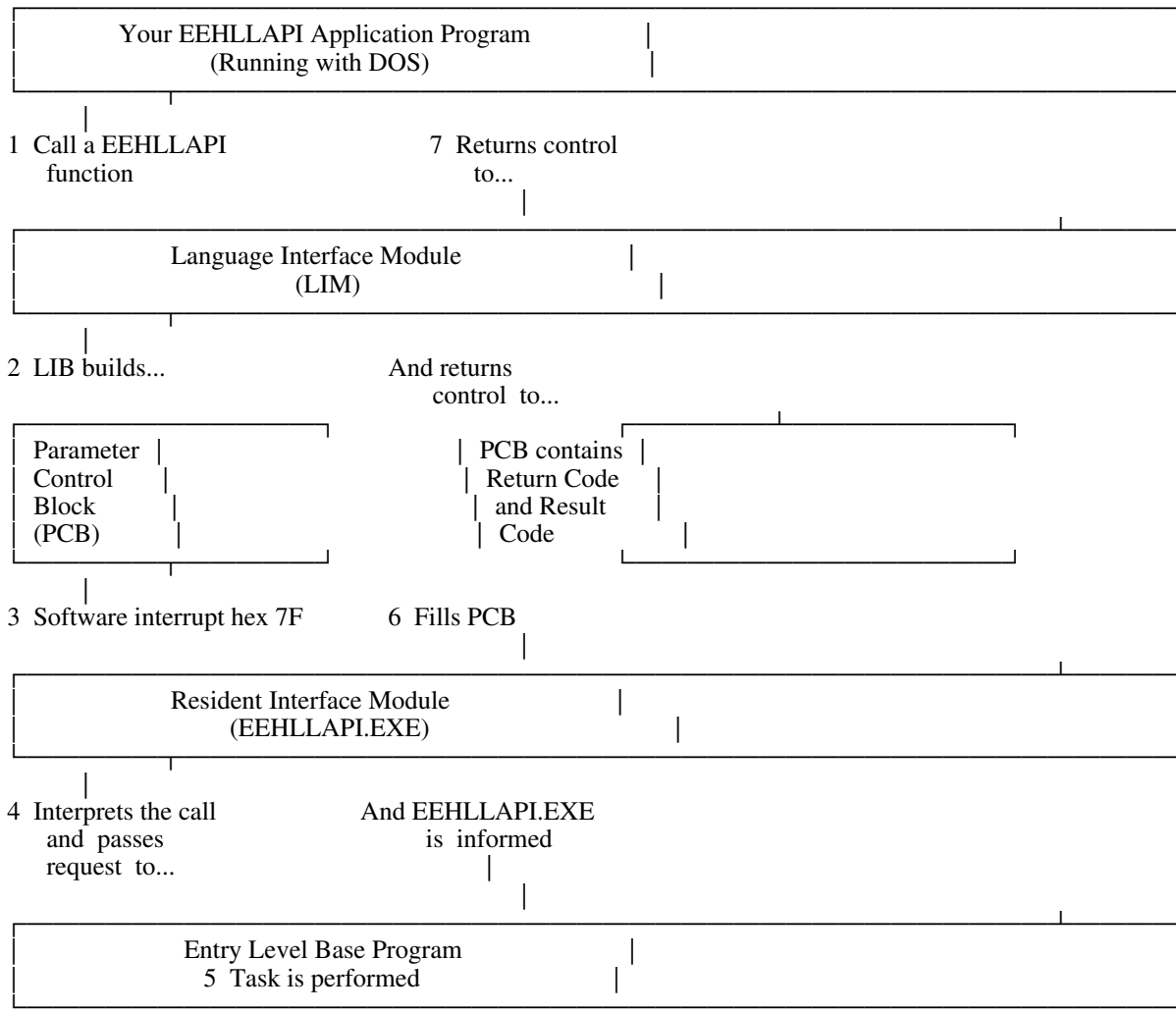

Figure 1. The Parts of EEHLLAPI

Functions a LIM Can Perform

The primary functions of a LIM are as follows:

Handle linkage from the calling application.

Obtain data parameter pointers or values.

Convert any language-unique data formats.

Construct a control block for the primary program.

Call (using an interrupt) EEHLLAPI.EXE. or AHLLAPI.EXE.

Receive control back from EEHLLAPI.EXE. or AHLLAPI.EXE.

Pass the necessary parameters and return control to the calling application.

If you are writing support for a language, you only need the above functions. If you want to extend these functions, there are two primary areas for interface extension:

1. Support for additional external functions
2. Modification or extensions of user calls to the primary program.

Frequently, users of Personal Computer languages develop multiple subroutine packages to provide utility functions such as display managers and data managers. Many of these packages require the application developers to call external subroutines. If you would like to consolidate several of your subroutines into one function package, you can write an extended LIM for this purpose.

Two different techniques are available:

1. The simpler technique is to reserve a range of function codes for each subroutine package. The LIM then examines the function code being invoked and calls the appropriate subroutine package. You might need to make adjustments in the number and type of parameters passed to EEHLLAPI.
2. A second technique for interface extensions is more complex. Function Code 0 has been reserved for a **Signal Gateway** function. In this mode, you can pass a “gatename” to your LIM. The interface can then use this function to establish routing for all subsequent function codes to a given subroutine.

   This technique is useful when existing subroutine packages have overlapping function codes. By switching all function requests through a specified gateway until a new “gateway” is requested, you can access a variety of external functions through a common call architecture.

Revised Call Support for the Personal Computer

The LIMs supplied by IBM are designed to provide a fixed-format, easy-to-use call format. You might want to tailor the number or format of the parameters to your specifications.

For example, calls to the interface use offsets into the host presentation space. As an ease-of-use feature, you might want to redesign the function calls to use a row/column format.

In this case your LIM would convert the row/column data into the offset values required by the actual interface function module.

Other revisions you might make to call the architecture could include:

Restrict certain functions from your application programmers.

Set up the session parameters automatically.

Do extended validity checking of function calls.

Make revisions in the call structure to fit your requirements.

Provide support for different data formats.

Details of Writing Your Own Language Interface Module

The LIM is responsible for building a parameter control block with the requisite data and issuing an Interrupt hex 7F to call the function module. Before issuing the Interrupt hex 7F, you must set register AX to hex 0104 and should set register BX to 0.

As good programming technique, you should verify that an interrupt vector has been established for the interface's EEHLLAPI.EXE module. The parameter control block has the following format:

```
PCB_HDR       DB          'PCB'    ;PCB header (in Caps)
PCB_FUNC      DB                   ;Function code VALUE
PCB_DSEG      DW                    ;String seg addr
PCB_DADDR     DW                     ;String offset addr
PCB_LENGTH    DW                     ;Data length VALUE
PCB_FILLER    DB                   ;Unused
PCB_RETCODE   DW                     ;Return code VALUE
PCB_STRLEN    DW           25      ;Maximum user string size
```

These rules apply in building the parameter control block:

The control block header PCB (in upper case) must appear at the start of the control block.

The function code must be the 1-byte numeric (binary) function code.

The pointer to the user's data string must contain the segment and offset address (note that the segment address precedes the offset).

The string pointer should point to the actual data string, and not to a length or pointer prefix.

The length should be the actual binary length value passed by the user. Since some function calls pass data in the fourth parameter (normally the return code), this value should be passed in the parameter control block.

The string length field contains the actual length of the user's string, if the language you are using to write your EEHLLAPI program provides a length value. This allows the code to avoid overlays by checking the length before placing data in the string. This field is only used by languages such as BASIC that provide the string length in the linkage data.

If your language does not provide the allocated length of a string, default this to 25000.

There are special considerations for Storage Manager calls. The internal interface to the storage manager is different from the regular EEHLLAPI interface. The storage manager shares the 7F interrupt vector, but does not use a PCB call. It has a register interface. The LIM service layer remaps the standard format user call into this register interface. The storage manager interface is:

```
INT 7FH
Registers: AX      1 4H (mandatory)
             BX     rrAAH (BL must = AAhex),
                        rr= 1- 4 request code
                        ( 1=get,  2=free,  4=freeall)
             CX     storage size (on GET request)
                    block ID (on FREE request)

Returns:
Call:      1 (get):      CX=block ID, DX=RETC
                                        AX=storage  seg
                                        BX=offset
           2  (free):                  DX=RETC
            4 (free all)               DX=RETC
```

Note: The storage manager uses a register interface, not the EEHLLAPI PCB. The support for Function 17 is handled at the LIM level. LIMs supplied by IBM support this call. User-supplied LIMs might not provide this feature.

Once the control block is built, call the resident module with the INT hex 7F interrupt. The DS:SI registers should point to the control block. You should save any required nonsegment registers. After executing the requested function, control is returned to your next sequential instruction after the INT hex 7F. The return code and any result codes are passed back to the parameter control block. You should pass these values back to the application caller's data area.

Note that EEHLLAPI passes the string data back directly, but the return code is passed back to the parameter control block, and you must retain the user's data addresses and pass back the return result codes. This allows you to examine and, if necessary, reformat these values.

Appendix C. OIA Image and Bit Group Information

This appendix explains Positions 2 through 81 and 82 through 103 in the host Operator Information Area (OIA). This information is vital to interpreting the returned data string under the **Copy OIA** (13) function.

OIA Image Group (Positions 2 through 81)

For CUT connection

The meaning of symbols in the OIA Image Group are understood by looking up the desired hexadecimal code in Figure 2. This figure shows the hexadecimal codes found in the host presentation space, and the characters they represent. Refer to *Personal Communications Version 4.1 Up and Running for DOS (Entry-Level Mode)* for more information on the OIA indicators .

For Home3270

The OIA image returned by the **Copy OIA** (13) function is in the ASCII format passed down by an attached protocol converter. The exact image of OIA depends upon the particular protocol converter used.

	0x	1x	2x	3x	4x	5x	6x	7x	8x	9x	Ax	Bx	Cx	Dx	Ex	Fx
x0	NUL	SP	0	&	à	ä	À	Ä	a	q	A	Q				
x1	EM	=	1		è	ë	È	Ë	b	r	B	R				
x2	FF	'	2	.	ì	ï	Ì	Ï	c	s	C	S				
x3	NL	"	3	,	ò	ö	Ò	Ö	d	t	D	T				
x4	STP	/	4	:	ù	ü	Ù	Ü	e	u	E	U				
x5	CR	\	5	+	ã	â	Ã	Â	f	v	F	V				
x6		|	6		õ	ê	Õ	Ê	g	w	G	W				
x7			7		ÿ	î	Y	Î	h	x	H	X				
x8	>	?	8	°	à	ô	A	Ô	i	y	I	Y				
x9	<	!	9		è	û	E	Û	j	z	J	Z				
xA	[	$		^	é	á	E	Á	k	æ	K	Æ				
xB	]	¢	§	~	ì	é	I	É	l	ø	L	Ø				
xC	)	£	#	¨	ò	í	O	Í	m	a	M	A				
xD	(	¥	@	`	ù	ó	U	Ó	n	ç	N	Ç				
xE	}	Pts	%	'	ü	ú	Y	Ú	o	;	O	;				
xF	{		_		ç	ñ	C	Ñ	p	*	P	*				Not Sup-ported

Figure 2. Host Presentation Space Character Table

© Copyright IBM Corp. 1996

OIA Group Indicator Meanings (Positions 82-103)

The states of each group are ordered so that the high order bits represent the indicators of higher priority. Therefore, if more than one state is active within a group, the state with the highest priority is the active state within that group.

Bytes 1 and 5 from Group 8 are used for the Entry Level. The remaining bytes and Groups are reserved.

Group 8: Input inhibited (5 bytes)

Byte 1:

Bit	Meaning
0-1	Reserved
2	Machine check
3	Communications check
4	Program check
5-7	Reserved

Byte 5:

Bit	Meaning
0	Reserved
1	Application program has operator input inhibited
2–7	Reserved

For Home3270, the OIA Group Indicator defined as above is returned when Entry Level is attached to the IBM 3174 with the AEA feature.

Appendix D. Sample Programs

This appendix contains code samples. The tables appearing before the sample programs list the functions used in the sample.

BASIC Sample Program

Function Name	Function Number
Connect	1
Copy Field to String	34
Disconnect	2
Find Field Length	32
Find Field Position	31
Receive	91
Reset System	21
Search Field	30
Sendkey	3
Wait	4

```
1  REM EEHLLAPI Version 1.XX
2  REM SAMPLE PROGRAM
3  REM (c) COPYRIGHT 1986,1987, IBM CORPORATION
4  KEY OFF: CLS

5  LOCATE 1,1  : PRINT "          EEHLLAPI SAMPLE BASIC PROGRAM"

6  LOCATE 2,1  : PRINT "(c)Copyright 1986,1987 IBM Corporation"
7  CLEAR
8  REM This sample basic program uses EEHLLAPI to provide an
9  REM interface to the user for downloading multiple files
1   REM from the host.
11  REM ------------------------------------------------------
12  REM     MANDATORY INT BASIC INITIALIZATION
13  REM ------------------------------------------------------
14  DEF SEG =  : APILOC=PEEK(&H1FE) + PEEK(&H1FF)

15  IF APILOC =      THEN PRINT "EEHLLAPI IS NOT AVAILABLE": END
16  DATA &H 4B8, &HB3 1, &HCDBB, &HCA7F, &H   2
17  DIM APICALL% (4)
18  FOR I=  TO 4: READ APICALL%(I): NEXT I
19  BLIMSET = VARPTR (APICALL%( ))
2   DEF SEG: CALL BLIMSET (APICALL%(1))
21  BLIM=APICALL%(1): DEF SEG = APICALL%( )

22  IF BLIM = &HB3 1 THEN PRINT "EEHLLAPI NOT AVAILABLE": END
23  REM ------------------------------------------------------
24  REM     END OF REQUIRED INIT. LOGIC
25  REM ------------------------------------------------------
26  GOTO 38
27  REM ------ Common routine to call EEHLLAPI ----------------
28  CALL BLIM (FUNC%,SDATA$,DLEN%,RETC%)
29  IF RETC% = 9 AND FUNC% <> 99 THEN GOTO 31
```

© Copyright IBM Corp. 1996

```
3   RETURN
31  REM --- System Error ---
32  CLS

33  PRINT "A System Error was detected during function:";FUNC%
34  FUNC%=2 : SDATA$ = SPACE$(35)
35  CALL BLIM (FUNC%,SDATA$,DLEN%,RETC%)

36  PRINT "Error component information ="; MID$(SDATA$,2 ,8)
37  END
38  REM ============================================================
39  REM ---------------------- MAIN PROGRAM --------------------
4   REM ============================================================
41  REM ============ RESET SYSTEM =============
42  FUNC% = 21
43  GOSUB 27
44  REM ----------------------------------------
45  CLS

46  LOCATE 5,5 : PRINT "Drive # (1..26) for RECEIVE.COM";

47  INPUT " : " ,DRIVE%

48  HOST$ = "E"
49  REM >>> Connect to Host Session
5   REM ================ CONNECT ================
51  FUNC% = 1
52  SDATA$ = HOST$
53  GOSUB 27
54  REM ----------------------------------------

55  IF RETC% <>   THEN PRINT "Could not connect to host" : END
56  REM >>> Clear Host Screen
57  REM ================ SENDKEY ================
58  FUNC% = 3

59  SDATA$ = "@C"
6   DLEN% = 2
61  GOSUB 27
62  REM ----------------------------------------
63  CLS
64  REM --------- PROMPT FOR FILE SPECS ---------
65  LOCATE 5,1

66  PRINT "Please input the info for files to be ";

67  PRINT "considered for downloading from the host."

68  PRINT "Use the wild card character ( ) in order ";

69  Print "to consider a group of files."

7   LOCATE 11,15: INPUT "filename : ",FILENAME$

71  LOCATE 13,15: INPUT "filetype : ",FILETYPE$

72  LOCATE 15,15: INPUT "filemode : ",FILEMODE$
```

```
73  FILELIST$ = FILENAME$ + " " + FILETYPE$ + " " + FILEMODE$
74  REM >>> Filelist Command
75  REM =============== SENDKEY ===============
76  FUNC% = 3

77  SDATA$ = "filelist " + fILELIST$ + "@E"
78  DLEN% = LEN(SDATA$)
79  GOSUB 27
8   REM ----------------------------------------
81  REM >>> Wait for Host to execute filelist command
82  REM ================ WAIT ================
83  FUNC% = 4
84  GOSUB 27
85  REM ----------------------------------------
86  REM >>> Determine position of Header Field
87  REM ========= FIND FIELD POSITION =========
88  FUNC% = 31

89  SDATA$ = "N "
9   RETC% = 1
91  GOSUB 27
92  REM ----------------------------------------
93  IF RETC% =   THEN GOTO 95

94  PRINT "Could not find position of header field" : END
95  FPOS% = DLEN%
96  IF FPOS% < 2   THEN GOTO 98

97  PRINT "Found bad header field position" : END
98  REM >>> Determine Size of Header Field
99  REM ========== FIND FIELD LENGTH ==========
1    FUNC% = 32

1 1  SDATA$ = "T "
1 2  RETC% = FPOS%
1 3  GOSUB 27
1 4  REM ----------------------------------------
1 5  FLEN% = DLEN%
1 6  REM >>> Search Header Field for 'Filename'
1 7  REM =============== SRCH FIELD =============
1 8  FUNC% = 3

1 9  SDATA$ = "Filename"
11   DLEN% = 8
111  RETC% = FPOS%
112  GOSUB 27
113  REM ----------------------------------------
114  IF RETC% <> 24 THEN GOTO 116

115  PRINT "Could not find header string" : END
116  REM >>> The file info (name,type,mode) in
117  REM >>> each field begins under 'Filename"
118  OFFSET% = DLEN% - FPOS%
119  CLS
12   LOCATE 4,1

121  PRINT "Please jump to your Host session and ";
```

```
122  PRINT "put an asterisk ( ) in the cmd field "

123  PRINT "for each file that you would like ";

124  PRINT "downloaded from the host.    The set "

125  PRINT "of files to be downloaded must include ";

126  PRINT "the first file and any number of "

127  PRINT "consecutive files after the first. ";

128  PRINT "Then jump back to your PC session and "

129  PRINT "press enter to continue . . .",X$
13   CLS
131  REM --- ============================ ---
132  REM --- BEGIN LOOP GET/TRANSFER FILE ---
133  REM --- ============================ ---
134  CLS
135  REM >>> Find the beginning of the next field
136  REM ========= FIND FIELD POSITION =========
137  FUNC% = 31

138  SDATA$ = "N "
139  RETC% = FPOS%
14   GOSUB 27
141  REM ----------------------------------------
142  IF RETC% =   THEN GOTO 144

143  PRINT "Could not find next field" : END
144  FPOS% = DLEN%

145  IF FPOS% > 1441 THEN PRINT "Bad Field" : END
146  REM >>> Search cmd field for flag
147  REM ============== SRCH FIELD ==============
148  FUNC% = 3

149  SDATA$ = " "
15   DLEN% = 1
151  RETC% = FPOS%
152  GOSUB 27
153  REM ----------------------------------------
154  REM >>> If no flag, then all done
155  IF RETC% = 24 THEN GOTO 215
156  REM >>> Copy file info from field
157  REM ========= COPY FIELD TO STRING =========
158  FUNC% = 34
159  DLEN% = FLEN%
16   SDATA$ = SPACE$(DLEN%)
161  RETC% = FPOS% + OFFSET%
162  GOSUB 27
163  REM----------------------------------------
164  HOSTFILE$ = MID$(SDATA$,OFFSET% + 1,2 )
165  REM --- remove spaces from file spec string ---

166  TEMP% = INSTR(HOSTFILE$," ")
167  IF TEMP% =   THEN GOTO 172
```

```
168  TEMP1$ = MID$(HOSTFILE$,1,TEMP$)
169  TEMP2$ = MID$(HOSTFILE$,TEMP% + 2,LEN(HOSTFILE$))
17   HOSTFILE$ = TEMP1$ + TEMP2$
171  GOT 166
172  REM --- Get PC File specs ---
173  CLS

174  LOCATE 5,5 : PRINT "DOWNLOADING :"
175  LOCATE 7,25: PRINT HOSTFILE$
176  LOCATE 1 ,5

177  PRINT "Please input PC file specifications."
178  LOCATE 11,5

179  PRINT "If you specify a path, be ";

18   PRINT "be sure that the path ends in -"

181  LOCATE 13,15: INPUT "path         : ",PCPATH$

182  LOCATE 15,15: INPUT "filename : ",PCNAME$

183  LOCATE 17,15: INPUT "filetype : ",PCTYPE$

184  PCFILE$ = PCPATH$ + PCNAME$ + "." + PCTYPE$

185  LOCATE 19,15: INPUT "options    : ", OPTIONS$

186  OPTIONS$ = "(" + OPTIONS$
187  REM >>> Must disconnect in order to call Function 91
188  REM ============== DISCONNECT ==============
189  FUNC% = 2
19   GOSUB 27
191  REM ---------------------------------------
192  CLS
193  LOCATE 1,1

194  PRINT "Downloading : ";HOSTFILE$;" to ";PCFILE$
195  LOCATE 3,1
196  REM ============== RECEIVE FILE ============
197  FUNC% = 91

198  SDATA$ = PCFILE$ = " " = HOSTFILE$ + " " + OPTIONS$
199  DLEN% = LEN(SDATA$)
2    RETC% = DRIVE%
2 1  GOSUB 27
2 2  REM ---------------------------------------

2 3  LOCATE 24,1  : INPUT "Press Enter to continue....",X$
2 4  REM >>> Reconnect to Host Session
2 5  REM ================ CONNECT ================
2 6  FUNC% = 1
2 7  SDATA$ = HOST$
2 8  GOSUB 27
2 9  REM ---------------------------------------
21   GOTO 131
211  REM --- ========================== ---
212  REM --- END LOOP GET/TRANSFER FILE ---
```

```
213  REM --- ========================== ---
214  REM >>> Get out of filelist screen
215  REM ================ SENDKEY ================
216  FUNC% = 3

217  SDATA$ = "@3"
218  DLEN% = 2
219  GOSUB 27
22   REM ----------------------------------------
221  REM >>> Reset System to known state (Disconnect, etc.)
222  REM ============ RESET SYSTEM ==============
223  FUNC% = 21
224  GOSUB 27
225  REM ----------------------------------------
226  CLS

227  LOCATE 2,5 : PRINT "DOWNLOADING IS COMPLETED"

228  LOCATE 5,5 : INPUT "(b) BASICA or (d) DOS ? ",SCONT$
229  CLS

23   IF SCONT$ = "b" THEN GOTO 233

231  IF SCONT$ = "d" THEN GOTO 234
232  GOTO 234
233  KEY ON : END
234  SYSTEM
235  ============================================================
```

COBOL Sample Program

```
IDENTIFICATION DIVISION
PROGRAM-ID. SAMPLER.
    This a cobol sampler program which allows
    the user to test EEHLLAPI function calls.
AUTHOR. PSC
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
====GLOBAL VARIABLES====
77   CHOICE                    PIC 999.
77   DLEN-DISPLAY               PIC 9999.
77   RETC-DISPLAY               PIC 9999.
----EEHLLAPI PARAMETERS----
77   FUNC                      PIC 99 COMP-
77   DLEN                      PIC 99 COMP-
77   RETC                      PIC 99 COMP-
77   DATA-STR                   PIC X(8 ) VALUE SPACES.
----MISC INPUT STRINGS----
77   PSID-STR                  PIC X VALUE SPACES.
77   EMPTY-STR                   PIC X(8 ) VALUE SPACES.
 1   HOST-EVENT-BUFF.
       3 HE-PSID                PIC X VALUE ' '.
       3 HE-TYPE                 PIC X VALUE 'B'.
       3 HE-PTR                  PIC X(4) VALUE LOW-VALUES.
      3  FILLER                  PIC  X(256).
============================================================
SCREEN SECTION.
 1   NUMER-MENU.
----NUMERICAL LIST OF FUNCTIONS----
       3 BLANK SCREEN.
       3 LINE    1 COLUMN 24 VALUE 'EEHLLAPI NUMERICAL MENU'.
       3 LINE    2 COLUMN 24 VALUE '======================'.
       3 LINE    3 COLUMN 1    VALUE ' 1 - Connect'.
       3 LINE    4 COLUMN 1    VALUE ' 2 - Disconnect'.
       3 LINE    5 COLUMN 1    VALUE ' 3 - Send Key'.
       3 LINE    6 COLUMN 1    VALUE ' 4 - Wait'.
       3 LINE    7 COLUMN 1    VALUE ' 6 - Search PS'.
       3 LINE    8 COLUMN 1    VALUE ' 7 - Q Cursor Location'.
       3 LINE    9 COLUMN 1    VALUE ' 8 - Copy PS to String'.
       3 LINE 1  COLUMN 1    VALUE ' 9 - Set Session Parms'.
       3 LINE 11 COLUMN 1    VALUE '1  - Q Sessions'.
       3 LINE    3 COLUMN 27 VALUE '11 - Reserve'.
       3 LINE    4 COLUMN 27 VALUE '12 - Release'.
       3 LINE    5 COLUMN 27 VALUE '13 - Copy OIA'.
       3 LINE    6 COLUMN 27 VALUE '14 - Q Field Attribute'.
       3 LINE    7 COLUMN 27 VALUE '15 - Copy String to PS'.
       3 LINE    8 COLUMN 27 VALUE '18 - Pause'.
       3 LINE    9 COLUMN 27 VALUE '2  - Q System".
       3 LINE 1  COLUMN 27 VALUE '21 - Reset System'.
       3 LINE 11 COLUMN 27 VALUE '22 - Q Session Status.'
       3 LINE    3 COLUMN 53 VALUE '23 - Start Host Notify'.
       3 LINE    4 COLUMN 53 VALUE '24 - Q Host Update'.
       3 LINE    5 COLUMN 53 VALUE '25 - Stop Host Notify'.
```

```
          3 LINE    6 COLUMN 53 VALUE '3  - Search Field'.
          3 LINE    7 COLUMN 53 VALUE '31 - Find Field Pos.'.
          3 LINE    8 COLUMN 53 VALUE '32 - Find Field Length'.
          3 LINE    9 COLUMN 53
              VALUE '33 - Copy String to Field'.
          3 LINE 1  COLUMN 53
              VALUE '34' - Copy Field to String'.
          3 LINE 11 COLUMN 53 VAKUE '99 - Convert Pos/RowCol'.
 1      CHOICE-SCR.
----SELECTION SCREEN----
      This screen is used to prompt the user for the
      desired function.    It also presents the user with
      choices for different menus and program termination.
          3    LINE 14 COLUMN 2  VALUE 'Select a function: '.
          3    LINE 14 COLUMN 39 PIC ZZ9 TO CHOICE.
          3    LINE 23 COLUMN    1 BLANK LINE.
          3    LINE 24 COLUMN 2  VALUE '3  -Exit Sample Program'.
----PROMPT SCREENS-----------------------------------------
 1    STRING-PROMPT.
      This screen is used to prompt the user for the
      Data String calling parameter.
          3    LINE 15 COLUMN 13 VALUE 'String : ".
          3    LINE 15 COLUMN 23 PIC X(55) TO DATA=STR.
 1    PSID-PROMPT.
      This screen is used to prompt the user for the
      Data String calling parameter when the Data String
      only needs a PSID.
          3    LINE 15 COLUMN 13 VALUE 'PSID : '.
          3    LINE 15 COLUMN 21 PIC X TO PSID-STR.
 1    HOST-STR-PROMPT.
      This screen is used to prompt the user for the
      PSID and option code for function 23, Start Host
     Notification.
          3    LINE 15 COLUMN 13 VALUE 'PSID : ".
          3    LINE 15 COLUMN 21 PIC X TO HE-PSID.
          3    LINE 15 COLUMN 25 VALUE 'option code : '.
          3    LINE 15 COLUMN 39 PIC X TO HE-TYPE.
 1    LENGTH-PROMPT.
      This screen is used to prompt the user for the
      Length calling parameter.
          3    LINE 16 COLUMN 13 VALUE 'Length : '.
          3    LINE 16 COLUMN 23 PIC ZZZ9 TO DLEN JUST.
 1    RC-PROMPT.
      This screen is used to prompt the user for the
      PS Position/Return Code calling parameter.
          3    LINE 17 COLUMN 13 VALUE 'PS Position : '.
          3    LINE 17 COLUMN 28 PIC ZZZ9 TO RETC JUST.
----DISPLAY SCREENS----------------------------------------
 1    STRING-DISPLAY.
      This screen displays the returned Data String.
          3    LINE 19 COLUMN 13 VALUE 'String : '.
          3    LINE 19 COLUMN 23 PIC X(55) FROM DATA-STR.
 1    LENGTH-DISPLAY.
      This screen displays the returned Length.
          3    LINE 2  COLUMN 13 VALUE 'Length : ".
          3    LINE 2  COLUMN 23 PIC ZZZ9 FROM DLEN-DISPLAY.
 1    RC-DISPLAY.
      This screen displays the returned Returned Code.
```

```
        3    LINE 21 COLUMN 13 VALUE 'Return Code : '.
        3    LINE 21 COLUMN 28 PIC ZZZ9 FROM RETC-DISPLAY.
----MISCELLANEOUS SCREENS-------------------------------
 1    BLANK-SCR.
        3 BLANK SCREEN.
 1    BLANK-LINE.
        3 BLANK LINE.
 1    CONT-SCR.
        3    LINE 23 COLUMN 23 BELL REVERSE-VIDEO
             VALUE 'Press Enter to continue'.
        3    LINE 24 COLUMN    1 BLANK LINE.
============================================================
PROCEDURE DIVISION.
MAIN-PARAGRAPH.
     This procedure displays a numerical list
     of the available functions and passes control
     to USERS-CHOICE for implementation of the
    desired  function(s).
       DISPLAY  NUMER-MENU.
        GO TO USERS-CHOICE.
USERS-CHOICE
     This procedure allows the user to test functions from
     the current menu.    It passes control for implementation
     of specific functions to the appropriate procedures.
       DISPLAY (14, 1) ERASE.
       MOVE   TO CHOICE.
      DISPLAY  CHOICE-SCR.
      ACCEPT  CHOICE-SCR.
     Does the user want to exit the program:
       IF CHOICE = 3   GO TO DONE.
----INITIALIZE VARIABLES---------------------------------
        MOVE CHOICE TO FUNC.
        MOVE   TO DLEN RETC.
        MOVE EMPTY-STR TO DATA-STR.
----PERFORM FUNCTION-------------------------------------
     This part of the procedure passes control to the
     appropriate procedure for implementing the desired
    EEHLLAPI  function.
       IF FUNC = 1    PERFORM CONNECT.
       IF FUNC = 2    PERFORM DISCONNECT.
       IF FUNC = 3    PERFORM SENDKEY.
       IF FUNC = 4    PERFORM WAIT.
       IF FUNC = 6    PERFORM SEARCH-PS.
       IF FUNC = 7    PERFORM Q-CURSOR-LOC.
       IF FUNC = 8    PERFORM COPY-PS-STRING.
       IF FUNC = 9    PERFORM SET-SESSION-PARMS.
       IF FUNC = 1  PERFORM QUERY-SESSIONS.
       IF FUNC = 11 PERFORM RESERVE-KEYBD.
       IF FUNC = 12 PERFORM RELEASE-KEYBD.
       IF FUNC = 13 PERFORM COPY-OIA.
       IF FUNC = 14 PERFORM Q-FIELD-ATTRIBUTE.
       IF FUNC = 15 PERFORM COPY-STRING-PS.
       IF FUNC = 18 PERFORM PAUSE.
       IF FUNC = 2  PERFORM Q-SYSTEM.
       IF FUNC = 21 PERFORM RESET-SYSTEM.
       IF FUNC = 22 PERFORM Q-SESS-STATUS.
       IF FUNC = 23 PERFORM START-HOST.
       IF FUNC = 24 PERFORM Q-HOST.
```

```
        IF FUNC = 25 PERFORM STOP-HOST.
        IF FUNC = 3  PERFORM SEARCH-FIELD.
        IF FUNC = 31 PERFORM FIND-FIELD-LENGTH-POS.
        IF FUNC = 32 PERFORM FIND-FIELD-LENGTH-POS.
        IF FUNC = 33 PERFORM COPY-STRING-FIELD.
        IF FUNC = 34 PERFORM COPY-FIELD-STRING.
        IF FUNC = 99 PERFORM CONVERT-POS-ROWCOL.
----DISPLAY RETURN CODE AND PAUSE------------------------
     After any EEHLLAPI function call, we want to see
     the Return Code and pause to examine the parameters
     passed and returned.
       DISPLAY  RC-DISPLAY.
       DISPLAY  CONT-SCR.
        DISPLAY (23, 1) '    '.
        STOP ' '.
        GO TO USERS CHOICE.
====EEHLLAPI FUNCTIONS==================================
     These procedures control the implementation of
     specific EEHLLAPI function calls.    Procedure calls
     are made to the miscellaneous IO procedures for the
     prompting and displaying of parameters.
CONNECT.
       PERFORM  GET-PSID.
       PERFORM  EEHLLAPI-CALL.
CONVERT-POS-ROWCOL.
       PERFORM  GET-STRING.
       PERFORM  GET-LENGTH.
       PERFORM  GET-RC.
       PERFORM  EEHLLAPI-CALL.
       DISPLAY  LENGTH-DISPLAY.
COPY-FIELD-STRING.
       PERFORM  GET-LENGTH.
       PERFORM  GET-RC.
       PERFORM  EEHLLAPI-CALL.
       DISPLAY  STRING-DISPLAY.
COPY-OIA.
       PERFORM  GET-LENGTH.
       PERFORM  EEHLLAPI-CALL.
       DISPLAY  STRING-DISPLAY.
COPY-PS-STRING.
       PERFORM  GET-LENGTH.
       PERFORM  GET-RC.
       PERFORM  EEHLLAPI-CALL.
       DISPLAY  STRING-DISPLAY.
COPY-STRING-FIELD.
       PERFORM  GET-STRING.
       PERFORM  GET-LENGTH.
       PERFORM  GET-RC.
       PERFORM  EEHLLAPI-CALL.
COPY-STRING-PS.
       PERFORM  GET-STRING.
       PERFORM  GET-LENGTH.
       PERFORM  GET-RC.
       PERFORM  EEHLLAPI-CALL.
DISCONNECT.
       PERFORM  EEHLLAPI-CALL.
FIND-FIELD-LENTH-POS.
       PERFORM  GET-STRING.
```

```
           PERFORM  GET-RC.
           PERFORM  EEHLLAPI-CALL.
           DISPLAY  LENGTH-DISPLAY.
PAUSE.
           PERFORM  GET-LENGTH.
           PERFORM  EEHLLAPI-CALL.
Q-CURSOR-LOC.
           PERFORM  EEHLLAPI-CALL.
           DISPLAY  LENGTH-DISPLAY.
Q-FIELD-ATTRIB.
           PERFORM  GET-RC.
           PERFORM  EEHLLAPI-CALL.
           DISPLAY  LENGTH-DISPLAY.
Q-HOST.
           PERFORM  GET-PSID.
           PERFORM  EEHLLAPI-CALL.
QUERY SESSIONS.
            MOVE 24 TO DLEN.
           PERFORM  EEHLLAPI-CALL.
           DISPLAY  LENGTH-DISPLAY.
           DISPLAY  STRING-DISPLAY.
Q-SESS-STATUS.
           PERFORM  GET-PSID.
            MOVE 18 TO DLEN.
           PERFORM  EEHLLAPI-CALL.
           DISPLAY  STRING-DISPLAY.
Q-SYSTEM.
           PERFORM  EEHLLAPI-CALL.
           DISPLAY  STRING-DISPLAY.
RELEASE-KEYBD.
           PERFORM  EEHLLAPI-CALL.
RESERVE-KEYBD.
           PERFORM  EEHLLAPI-CALL.
RESET-SYSTEM.
           PERFORM  EEHLLAPI-CALL.
SEARCH-FIELD.
           PERFORM  GET-STRING.
           PERFORM  GET-LENGTH.
           PERFORM  GET-RC.
           PERFORM  EEHLLAPI-CALL.
           DISPLAY  LENGTH-DISPLAY.
SEARCH-PS.
           PERFORM  GET-STRING.
           PERFORM  GET-LENGTH.
           PERFORM  EEHLLAPI-CALL.
           DISPLAY  LENGTH-DISPLAY.
SENDKEY.
           PERFORM  GET-STRING.
           PERFORM  GET-LENGTH.
           PERFORM  EEHLLAPI-CALL.
SET-SESSION-PARMS.
           PERFORM  GET-STRING.
           PERFORM  GET-LENGTH.
           PERFORM  EEHLLAPI-CALL.
START-HOST.
           PERFORM  GET-HOST-STR.
            MOVE LOW-VALUES TO HE-PTR.
            MOVE 256 TO DLEN.
```

```
       CALL 'COBLIM' USING FUNC HOST-EVENT-BUFF DLEN RETC.
STOP-HOST.
      PERFORM  GET-PSID.
      PERFORM  EEHLLAPI-CALL.
WAIT.
      PERFORM  EEHLLAPI-CALL.
=============================================================
====GENERIC EEHLLAPI CALL=================================
    This is the EEHLLAPI function call used by most of
    the above procedures.    This procedure also moves the
    returned parameters into variables used for display.
EEHLLAPI-CALL.
       CALL 'HLLCOB' USING FUNC DATA-STR DLEN RETC.
       MOVE DLEN TO DLEN-DISPLAY.
       MOVE RETC TO RETC-DISPLAY.
====MISCELLANEOUS ROUTINES================================
GET-STRING.
    This procedure prompts the user for the Data String.
      DISPLAY  STRING-PROMPT.
      ACCEPT  STRING-PROMPT.
GET-PSID.
    This procedure prompts the user for the Data String
    in the case where the Data String requires only a PSID.
      DISPLAY  PSID-PROMPT.
      ACCEPT  PSID-PROMPT.
       MOVE PSID-STR TO DATA-STR.
GET-HOST-STR.
    This procedure prompts the user for the Data String
    needed for function 23, Start Host Notification.
      DISPLAY  HOST-STR-PROMPT.
      ACCEPT  HOST-STR-PROMPT.
GET-LENGTH.
    This procedure prompts the user for the Length.
      DISPLAY  LENGTH-PROMPT.
      ACCEPT  LENGTH-PROMPT.
GET-RC.
    This procedure prompts the user for the PS Position.
      DISPLAY  RC-PROMPT.
      ACCEPT  RC-PROMPT.
DONE.
    This procedure clears the screen and
    terminates the program.
      DISPLAY  BLANK-SCR.
      STOP  RUN.
```

Pascal Sample Program

Function Name	Function Number
Connect	1
Copy Presentation Space to String	8
Disconnect	2
Query Sessions	10
Query System	20
Release	12
Reserve	11
Reset System	21
Search Presentation Space	9
Sendkey	3
Wait	4

```
program set_time ( input, output);

( ========================================================== )
(  ---------- SAMPLE EEHLLAPI PROGRAM ----------  )
(    This program is designed to obtain the time and date      )
(    information from the host session and pass this            )
(    information to the PC session.
( ========================================================== )



const
  ( ---------------------------------------------------- )
  (   KEYBOARD  MNEMONICS
  (   These  mnemonics  are  only  valid  as  long  as  the  escape     )
  (   character  is  not  changed  from  its  default  value  of  '@'  )
  (   with  a  call  to  Set  Session  Parameters.
  ( ---------------------------------------------------- )

    clear_key           = '@C';
    enter_key           = '@E';
    reset_key           = '@R';



type
    ( ------------------------------------- )
     (  date and time record types for passing  )
     (  information to PC Session                    )
    ( ------------------------------------- )
     time_record = record
         hour  ,
         min  ,
         sec  ,
          tick : integer;
    end;
     date_record = record
         year  ,
         month  ,
          day : integer;
```

```
    end;


var
    (  EEHLLAPI calling parameters :     )
    Function_Number : integer;
    Data_String : string(144);
    Length : integer;
    Return_Code : integer;


    (  --- strings containing time and date information ---  )
    time_string,                (  time : hh:mm:ss                  )
    date_string : lstring(8);     (  date : mm/dd/yy                  )

    (  --- records containing time and date information ---  )
    time : time_record;
    date : date_record;

    (  --- PSID of Host Session determined during init    ---  )
    Host_Session : char;



(  External reference to call EEHLLAPI Pascal LIM  )
procedure HLLPAS ( var lim_func : integer;
                           var lim_str : string;
                          var  lim_len,
                           lim_retc : integer ); EXTERN;

( ---------------------------------------------------------- )
(     External references to Pascal library functions for     )
(     setting the time and date.    See the Pascal Compiler     )
(     Language Reference for more information on how to        )
(     use these functions.
( ---------------------------------------------------------- )
function setdat (year,month,day : integer)
                       : boolean; EXTERN;
function settim (hours,minutes,seconds,hundreds : integer)
                       : boolean; EXTERN;
( ---------------------------------------------------------- )
(     External reference to Pascal library procedure for     )
(     terminating the program.    See the Pascal Compiler       )
(     Language Reference for more infomation on how to         )
(     use this functions.                                        )
procedure ENDXQQ; EXTERN;



procedure check_system;
( ---------------------------------------------------------- )
(  This procedure should be called after any call to EEHLLAPI.  )
(  This procedure determines whether or not there was a system  )
(  error (return code of 9 from EEHLLAPI).    If a system error     )
(  did occur, then system information is obtained through a         )
(  call to Query System and program execution is halted.              )
```

```
( ------------------------------------------------------------ )
var
    i : integer;
begin
    if Return_Code = 9 then begin
        writeln('There was a system error.');
        writeln('Calling Query System to get system info.');

      (  ======== Query System ========  )
      Function_Number := 2 ;
      HLLPAS ( Function_Number,
                  Data_String,
                  Length,
                  Return_Code  );
      if Return_Code = 9 then
          writeln('Unrecoverable System Error.')
      else  begin
            writeln('<<< SYSTEM DATA >>>');
            write('                  Bytes 1 to 15 : ');
            for i := 1 to 15 do write(Data_String{i});
            writeln;
            write('                  Extended Error Code 1 : ');
            for i := 2  to 23 do write(Data_String{i});
            writeln;
            write('                  Extended Error Code 2 : ');
            for i := 24 to 27 do write(Data_String{i});
            writeln;
        end;   (  of else  )
        (  stop execution and return to DOS  )
        ENDXQQ;
   end
end;



procedure set_Host_Session;
( ------------------------------------------------------------ )
(  This procedure calls the EEHLLAPI function Query Sessions     )
(  in order to determine the short name of the host session.     )
(  The global variable 'Host_Session' is set accordingly.          )
( ------------------------------------------------------------ )
var
    offset,                          (  offset to the beginning of the  )
                                        (  12 bytes of info per session;     )
    i : integer;                   (  loop control variable                  )

begin
    (  ======= Query Sessions =======
    Function_Number := 1 ;
    Length := 144;
    HLLPAS ( Function_Number,
                Data_String,
                Length,
                Return_Code  );
   check_system;
    (  Note : # of sessions is returned in Length  )
    (  check each session to see if it is the host session  )
    for i := 1 to Length do begin
```

```
        (  calculate offset to ith session  )
        offset := (i – 1)   12 ;
        (  session type is returned in 1 th position  )
        if Data_String{offset+1 } = 'H' then
            Host-Session := Data_String{offset+1};
    end;    (  of for loop  )
end;



function digit ( ch : char ) : integer;
( ------------------------------------------------------- )
(     This function converts a digit in character form to     )
(     a digit in integer form.                                          )
( ------------------------------------------------------- )
begin
    digit := ord(ch) – ord(' ') ;
end;



procedure convert_info;
( ------------------------------------------------------------ )
(  This procedure is used to convert the time and date strings  )
(  into integer values so that they may be passed to DOS using  )
(  the Pascal function calls 'settim', and 'setdat'.    The             )
(  values are stored in records, the global variables 'time'       )
(  and 'date'.                                                                        )
( ------------------------------------------------------------ )
begin
    time.hour := 1    digit(time_string[1]) +
                            digit(time_string[2])  ;
    time.min := 1    digit(time_string[4]) +
                              digit(time_string[5])  ;
    time.sec := 1    digit(time_string[7]) +
                              digit(time_string[8])  ;
    time.tick :=  ;
    date.month := 1    digit(date_string[1]) +
                              digit(date_string[2])  ;
    date.day := 1    digit(date_string[4]) +
                            digit(date_string[5])  ;
    date.year := 1    digit(date_string[7]) +
                              digit(date_string[8])  +
                              19  ;
end;



procedure get_info ( const delimiter : lstring ;
                              var info_string : string );
( ------------------------------------------------------- )
(  This procedure searches the host screen for the given          )
(  delimiter.    If the delimiter is found, then the string of  )
(  of info ( time or date ) is copied into the given string     )
(  variable.
( ------------------------------------------------------- )
var
```

```
    delimiter_position : integer;

begin
    (  = Search Presentation Space ==  )
    Function_Number := 6;
   copystr(delimiter,Data_string);
    Length := 1;
    Return_Code := 1;
    HLLPAS ( Function_Number,
                Data_String,
                Length,
                Return_Code  );
   check_system;

  (  Note : position of string is returned in Length  )
  delimiter_position := Length;
  (  If the delimiter was found then copy the info  )
  if delimiter_position >   then begin
  (  ====== Copy PS to String =====  )
      Function_Number := 8;
      Length := 8;
     Return_Code  :=
              delimiter_position – 2;
      HLLPAS ( Function_Number,
                  info_string,
                  Length,
                  Return_Code  );
   check_system;

    end;   (  of if  )
end;



procedure query_for_info;
( ------------------------------------------------------------ )
(  This procedure attempts to query the host for the time.     )
( ------------------------------------------------------------ )
var
    cursor_location : integer;
begin
    (  ========== Send Key ==========  )
    Function_Number := 3;
   copystr(clear_key,  Data_String);
   Length  :=2;
    HLLPAS ( Function_Number,
                Data_String,
                Length,
                Return_Code  );
   check_system;

    (  =========== Wait ===========  )
    Function_Number := 4;
   copystr(clear-key,Data-string);
    Length := 2;
    HLLPAS ( Function_Number,
                Data_String,
                Length,
```

```
                Return_Code  );
    check_system;

    (  =====Search Presentation Space=====  )
    Function_Number := 6;
    copystr('RUNNING',Data_String);
    Length := 7;
    Return _Code :=1;
    HLLPAS ( Function_Number,
                Data_String,
                Length,
                Return_Code  );
    check_system;

    (  ========== Send Key ===========  )
    Function_Number := 3;
    copystr('query time ' enter_key,Data_String);
    Length := 12;
    HLLPAS ( Function_Number,
                Data_String,
                Length,
                Return_Code  );
    check_system;

    (  ============ Wait ============  )
    Function_Number := 4;
    HLLPAS ( Function_Number,
                Data_String,
                Length,
                Return_Code  );
    check_system;

    (  =====Search Presentation Space=====  )
    Function_Number := 6;
    copystr('RUNNING',Data_String);
    Length := 7;
    Return_Code := 1;
    HLLPAS ( Function_Number,
                Data_String,
                Length,
                Return_Code  );
    check_system;

    get_info ( ':' , time_string );
    get_info ( '/' , date_string );

end;



procedure get_time_from_host;
( ------------------------------------------------------------ )
(  This procedure calls 'query_for_info' to obtain the time        )
(  and date info from the host after it takes control of the     )
(  Host Session.                                                                    )
( ------------------------------------------------------------ )
begin
```

```
    set_Host_Session;

    (  Connect to Host and reserve keyboard  )
    (  ========== Connect ===========  )
    Function_Number := 1;
    copystr(Host_Session,Data_String);
    Length := 1;
    HLLPAS ( Function_Number,
                Data_String,
                Length,
                Return_Code  );
    check_system;

    (  ========== Reserve ===========  )
    Function_Number := 11;
    HLLPAS ( Function_Number,
                Data_String,
                Length,
                Return_Code  );
    check_system;

    query_for_info;

    (  ========== Release ============ )
    Function_Number := 12;
    HLLPAS ( Function_Number,
                Data_String,
                Length,
                Return_Code  );
    check_system;

    (  ========= Disconnect ========== )
    Function_Number := 2;
    HLLPAS ( Function_Number,
                Data_String,
                Length,
                Return_Code  );
    check_system;

end;



procedure give_time_to_PC;
( ------------------------------------------------------------ )
(  This procedure converts the time and date strings into        )
(  the integer form that is passed to the Pascal settim and     )
(  setdat functions.    If the functions return a false value     )
(  then a message is printed warning the user of the invalid  )
(  attempt to set the date and/or time.                                  )
( ------------------------------------------------------------ )
begin
    convert_info;
     if not settim(time.hour,time.min,time.sec,time.tick) then
          writeln('Time not set.');
     if not setdat(date.year,date.month,date.day) then
          writeln('Date not set.');
end;
```

```
(    MAIN PROGRAM  )

begin

    (  ======== Reset System ========  )
    Function_Number := 21;
    HLLPAS ( Function_Number,
                Data_String,
                Length,
                Return_Code  );
   check_system;

   get_time_from_host;

   give_time_to_PC;

    (  ======== Reset System ========  )
    Function_Number := 21;
    HLLPAS ( Function_Number,
                Data_String,
                Length,
                Return_Code  );
   check_system;

end.
```

“C” Sample Program

Function Name	Function Number
Connect	1
Convert Position or RowCol	99
Disconnect	2
Query Cursor Location	7
Query Field Attribute	14
Release	12
Reserve	11
Reset System	21

```
/                                        /
/  This sample C program is designed to use HLLAPI  /
/  to obtain position and attribute information of  /
/  the cursor location.    The user is prompted to      /
/  position the cursor accordingly, and is given       /
/  the cursor position and a decoding of the field  /
/  attribute byte at that position.                     /
/                                        /

/  Include statement for standard input/output module  /
#include <c:\ibmc\include\stdio.h>


/  HLLAPI Parameters  /
int       API_FUNC, API_LEN, API_RETC;
char      API_STRING[255];



unsigned int shift (byte,numbits)
/  This function returns the unsigned integer that  /
/  results from shifting byte right numbits.    If a  /
/  negative number is passed throught the numbits     /
/  parameter, byte is shifted left -numbits.              /
     unsigned int byte;
    int                  numbits;
    {
      unsigned int result;
      if (numbits >  )          /  right shift  /
           result = byte >> numbits;
      else                                /  left shift     /
           result = byte << -numbits;
     return(result);
      }



void byte_to_bits (byte,bits)
/  This function is designed to extract the 8 'bits' that  /
/     define 'byte' and return them in the 'bits' array.         /
     unsigned int byte;
     short int bits[8]
    {
```

```
    unsigned int andbyte;
   int  i;
    unsigned int tempbyte;
    for ( i = 7, andbyte = 1;    i >=  ; --i, andbyte  = 2 )
        {
        /  zero out all but ith bit  /
        tempbyte = byte & andbyte;
        /  shift ith bit to rightmost bit position  /
        bits[i] = shift(tempbyte,7-i);
        }
  }


void get_info ()
/  This function obtains information regarding  /
/  the attributes of the cursor location.            /
  {
   /  cursor location info  /
  int  cursor_pos,
       cursor_row,
       cursor_col,
       attribyte;


   /  Connect to Host Session  /
   API_FUNC = 1;

  strcpy(API_STRING,"E\O");
   API_LEN = 1;
  hllc(&API_FUNC,API_STRING,&API_LEN,&API_RETC);
   if (API_RETC !=  )
      {

        printf("Warning : could not connect to Host.\n");

        printf("Return Code : %d\n",API_RETC);
       return;
      };

   /  Reserve Keyboard  /
   API_FUNC = 11;
  hllc(&API_FUNC,API_STRING,&API_LEN,&API_RETC);


  /                    /
   /  Query Cursor Location  /
  /                    /
   API_FUNC = 7;
  hllc(&API_FUNC,API_STRING,&API_LEN,&API_RETC);
   if (API_RETC ==  )
      {
        cursor_pos = API_LEN;

        printf("Curosr Location : Position %d\n",cursor_pos);
      }
  else
      {
```

```
          printf("Warning : ",
                    "could not determine cursor location.\n");
          printf("Return Code : %d\n",API_RETC);
          return;
        };

/                          /
 /  Convert Position to RowCol  /
/                          /
 API_FUNC = 99;

strcpy(API_STRING,"EP\ ");
 API_RETC = cursor_pos;
hllc(&API_FUNC,API_STRING,&API_LEN,&API_RETC);
 if (API_RETC ==   || API_LEN ==  )
     {

        printf("Warning : Incorrect input error.\n");
       return;
     }
 else if (API_RETC == 9998)
     {

        printf("Warning : ",
                  "Invalid Host id or Host never connected.\n");
       return;
     }
 else if (API_RETC == 9999)
     {

        printf("Warning : input error in Data String\n");
       return;
     }
else
     {
        cursor_row = API_LEN;          cursor_col = API_RETC;

        printf("Cursor Location : Row %d, Column %d\n",
                   cursor_row,cursor_col);
     }

/                       /
 /  Query Field Attribute  /
/                       /
 API_FUNC = 14
 API_RETC = cursor_pos;
hllc(&API_FUNC,API_STRING,&API_LEN,&API_RETC);
 if (API_LEN ==  )
     {

        printf("The screen is unformatted.\n");
       return;
     }
 else if (API_RETC !=  )
     {

        printf("Could not determine the attribute byte.\n");
       return;
```

```
    }
else
    {
     int  i;
     /  attribute byte  /
     unsigned int attribyte  /
     /  bit breakdown of attribute byte  /
     short int attribits[8];
     short int attribit45;

     /  attribute byte is returned in length parameter  /
     attribyte = API_LEN;

     printf("Attribute byte (hex code) : %x\n",attribyte);
     /  break down attribyte into attribits  /
    byte_to_bits  (attribyte,attribits);

     printf("Attribute byte (bin code) : ");
     for ( i =  ; i <= 7; ++i )

          printf("%d",attribits[i]);

    printf("\n");


     /  Unprotected/Protected bit  /
     if (attribits[2] == 1)

     printf("Protected data field.\n");

      else

           printf("          Unprotected data field.\n");


      /  Alpha/Numeric bit  /
      if (attribits[3] == 1)

           printf("          Numeric data only.\n");
     else

           printf("          Alphanumeric data only.\n");

      /  Automatic skip  /
      if (attribits[2] ==1 && attribits[3] == 1)

          printf("          Automatic  skip.\n");

      /  Intensity/Selector Pen Detectability bit  /
      attribit45 = (2   attribits[4] + attribits[5];
      if (attribit45 ==   )
        {

          printf("          Normal  intensity.\n");

           printf("          Not pen detectable.\n");
        }
      else if (attribit45 == 1)
```

```
            {

             printf("          Normal  intensity.\n");

             printf("          Pen  detectable.\n");
            }
          else if (attribit45 == 2)
            {

             printf("          High  intensity.\n");

             printf("          Pen  detectable.\n");
            }
          else if (attribit45 == 3)
            {

             printf("          Non-display.\n");

              printf("          Not pen detectable.\n");
            }

        }

     / Release Keyboard /
     API_FUNC = 12;
    hllc(&API_FUNC,API_STRING,&API_LEN,&API_RETC);

     / Disconnect from Host Session /
     API_FUNC = 2;
    hllc(&API_FUNC,API_STRING,&API_LEN,&API_RETC);

   } / end of function get-info /



main ()
   {
    char ch;

    / Reset System /
    API_FUNC = 21;
   hllc(&API_FUNC,API_STRING,&API_LEN,&API_RETC);

   do {

       printf("\nPlease jump to your Host Session and move \n");

       printf("the cursor to the desired location.\n");

       printf("Press 'i<RETURN>' to obtain info, or\n");

      printf("          'q<RETURN>' to quit program.   -->");
       ch = getchar();
       getchar(); if (ch!='/ 12') getchar();
       if (ch == 'i') get_info();
     }
    while (ch != 'q');
```

```
  /  Reset System  /
   API_FUNC = 21
 hllc(&API_FUNC,API_STRING,&API_LEN,&API_RETC);
}
```

Appendix E. Alternate Code Page Support

EEHLLAPI, using DOS 3.30, has Code Page Support for the following countries:

Table 1. Supported Code Pages

Country Code	Supported Code Pages
BE	437, 850
BI	437, 850
BR	850, 437
CF	863, 850
DK	865, 850
FR	437, 850
GR	437, 850
IT	437, 850
LA	437, 850
NL	437, 850
NO	865, 850
PO	860, 850
SP	437, 850
SO	437, 850
SV	437, 850
SU	437, 850
SF	437, 850
SG	437, 850
UK	437, 850
US	437, 850

Note: If you configure DOS for a language that does not match your entry emulator language, the results of an ASCII translation will be unpredictable.

© Copyright IBM Corp. 1996

Index

A

AHLLAPI function numbers vii
AHLLAPI Functions 61
AHLLAPI.EXE
alternate presentation spaces
  functions requiring special attention 45
ASC3270.COM
Assembler 8088 language 76
  LIMs
  linking to
    Assembler 8088 76
  linking your EEHLLAPI program with 76
attribute bytes 14, 15, 18, 24, 48

B

BASIC language
  examples of 6
  functions not available to 69
  limitations because of 33, 52, 55
    solutions to 55

C

COBOL language 33, 52, 55, 72
  limitations because of 33, 52, 55
  linking your EEHLLAPI program with 72
Code Page Support 115
code, function vii
communication services functions
  Receive File 30
  Send File 41
Compiled BASIC 71
  linking your EEHLLAPI program with 71
  See also "BASIC language" 71
Connect Presentation Space 9, 10
  functions where not required 10
Copy Field to String 12
Copy Field to String – AHLLAPI 61
copy functions
  Copy Field to String 12
  Copy OIA 13
  Copy Presentation Space 14
  Copy Presentation Space to String 15
  Copy String to Field 17
  Copy String to Presentation Space 18
  parameters affecting 48
Copy OIA 13
Copy OIA – AHLLAPI 61
Copy Presentation Space 14
Copy Presentation Space to String 15
Copy Presentation Space to String – AHLLAPI 62
Copy Presentation Space – AHLLAPI 62
Copy String to Field 17
Copy String to Field – AHLLAPI 62
Copy String to Presentation Space 18
Copy String to Presentation Space – AHLLAPI 62

D

data length (calling) 7
  defined 7
data length (returned) 8
  defined 8
data string (calling) 7
  defined 7
data string (returned) 8
  defined 8
device services functions
  Release 34
  Reserve 35
Disconnect Presentation Space 19
Disconnect Presentation Space – AHLLAPI 62
Drop Session DTR – AHLLAPI 66

E

EAB 51
EEHLLAPI
  compiling your EEHLLAPI program 69
  messages 79
  Programs 89
  running your program 77
EEHLLAPI function numbers vii
EEHLLAPI Functions
  defined 7
EEHLLAPI.EXE
  defined 2
  loading 5
EHLBASM
  functions associated with
    Free All Storage subfunction 58
EHLSAMP
  See "Program Sampler" 5
Entry HLLAPI
  advantages of using 1
  defined 1
  loading the program 5
    from a diskette 5
  overview 1, 2
  parts of 2

© Copyright IBM Corp. 1996

F

field-related functions
  Copy Field to String 12
  Copy String to Field 17
  Find Field Length 20
  Find Field Position 21
  Search Field 37
file transfer 30, 41
  considerations for using 33
  Receive File function 30
  Send File function 41
Find Field Length 20
Find Field Length – AHLLAPI 63
Find Field Position 21
Find Field Position – AHLLAPI 63
Free All Storage subfunction 58
Free Storage subfunction 57
Function Calls 2
  defined 3
  notes on using the function 8
  page layout conventions 7
  parameters when called 7
  parameters when returned 8
  services provided 2
  use of 7
Function code
  numbers vii
  values vii
function code (calling) 8
  defined 8
function code (returned) 8
  defined 8
function numbers vii
functions, by number vii

G

Get Storage subfunction 56

H

hardware requirements xiii
HLLBASM 52, 53
  functions associated with 55
host requirements xiv

I

IBM C language
  linking your EEHLLAPI program with 76
Interpretive BASIC 69
  initialization code for EEHLLAPI 69
  limitations 14
  linking your EEHLLAPI program with 69
  restrictions 70
  See also "BASIC language" 69
Interpretive BASIC *(continued)*
  string manipulation 70

K

keyboard mnemonics 46
  See also "mnemonics" 46

L

language interface modules 2
  See LIMs. 2
languages supported xiii
licensing agreement ix
LIMs 2
  defined 3
  functions performed by 83
  languages supported 3
  linking to 69
    BASIC 69
    COBOL 72
    Compiled BASIC 71
    IBM C language 76
    Interpretive BASIC 69
    Pascal 73
  revised call support for the 3270 PC4 84
  user-generated 81, 85

M

messages 79
mnemonics 18, 46
  for Send Key 47

N

NOEAB 51
notices ix
NOXLATE 51
numbers, functions vii

O

OIA 87
Operator Information Area
  See "OIA" 13
operator services
  functions
    Pause 22
    Query Host Update 25
    Query Session Status 26
    Query Sessions 27
    Query System 28
    Send Key 45
    Set Session Parameters 48
    Start Host Notification 52
    Stop Host Notification 36, 54
    Wait 59

P

parameters when called 7
  defined 7
Pascal Language 73
  creating record overlays 74
  example 74
  limitations because of 33, 52, 55
  linking your EEHLLAPI program with 73
Pause 22
  parameters affecting 49
presentation services
  functions
    Connect Presentation Space 9
    Copy Field to String 12
    Copy OIA 13
    Copy Presentation Space 14
    Copy Presentation Space to String 15
    Copy String to Field 17
    Copy String to Presentation Space 18
    Disconnect Presentation Space 19
    Find Field Length 20
    Find Field Position 21
    Query Cursor Location 23
    Query Field Attribute 24
    Search Field 37
    Search Presentation Space 39
presentation space
  character table 87
  field-formatted 12, 17, 20, 21, 37
  related publications xiv
program operator
  defined 1
  tasks 1
program sampler
  installing and running 6

Q

Query Cursor Location 23
Query Emulator Host Connection – AHLLAPI 66
Query Field Attribute 24
Query Field Attribute – AHLLAPI 63
Query Host Update 25
Query Session Status 26
Query Sessions 27
Query System 28
Query System – AHLLAPI 63

R

Raise Session DTR – AHLLAPI 67
Receive File 30, 33
  considerations for using 33
Release 34
Reserve 35
resident interface module
  EEHLLAPI.EXE 2
  loading 5
return code 8
  defined 8

S

Samples 89
Search Field 37
Search Field – AHLLAPI 64
search functions
  parameters affecting 48
  Search Field 37
  Search Presentation Space 39
Search Presentation Space 39
Send File 33, 41
  considerations for using 33
Send Key 45
Send Key – AHLLAPI 64
sending keystrokes
  mnemonics 46
  Send Key function 45
Set Session Parameters 48
Set Session Parameters – AHLLAPI 65
software interrupts 76
software requirements xiii
specifying strings 18
Start Host Notification 52
Stop Host Notification 36, 54, 55
  functions
    Storage Manager 55
Storage Manager 55, 56, 57, 58
  subfunctions
    Free All Storage 58
    Free Storage 57
    Get Storage 56
string specification 48

T

trace 49, 77

W

Wait 49

X

XLATE 51

IBM®

Part Number: 39H0468

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

39H 468

SC31-8267-