IBM PC 3270 Emulation Program
Entry Level Version 1.21

Programmer's Guide

High Level Language
Application Program Interface

Communications Family

74X9879

IBM PC 3270 Emulation Program
Entry Level Version 1.21

Programmer's Guide

High Level Language
Application Program Interface

Communications Family

Third Edition (December 1987)

This book describes how to use the IBM PC 3270 Emulation Program, Entry Level, High-Level Language Application Program Interface.

References in this publication to IBM products, programs, or services do not imply that IBM intends to make them available in all countries in which IBM operates. Any reference to an IBM program product in this publication is not intended to state or imply that only IBM's program product may be used. Any functionally equivalent program may be used instead.

Publications are not stocked at the address given below. Requests for this or other IBM publications should be made directly to the IBM branch office serving your locality.

A Reader's Comments Form is provided at the back of this publication. If the form has been removed, address comments to IBM Corporation, Department 95H, 11400 Burnet Road, Austin, Texas 78758. IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Preface

This book provides programming information you will need to use the IBM Personal Computer 3270 Entry Emulator High-Level Language Application Program Interface (EEHLLAPI).

EEHLLAPI is used with the IBM PC 3270 Emulation Program, Entry Level (hereafter referred to as the Entry Level Emulation Program).

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

If you just want to get started using the Entry Level Emulation Program or have no interest in using the programming interface, see the *IBM PC 3270 Emulation Program, Entry Level, User's Guide.*

How This Book Is Organized

- Chapter 1, "Introduction," provides an overview of EEHLLAPI and discusses parts of the interface.
- Chapter 2, "Loading the Program," describes how to load the resident portion of EEHLLAPI and run the EEHLLAPI program sampler.
- Chapter 3, "EEHLLAPI Functions," describes each function in detail. These functions are listed in alphabetical order.
- Chapter 4, "Compiling and Running Your EEHLLAPI Application Program," provides information about linking your EEHLLAPI program with the appropriate Language Interface Module (LIM), running your program, and using Trace.
- Appendix A, "Messages," explains the messages you may see while using EEHLLAPI.
- 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 Emulator diskette.
- Appendix C, "OIA Image and Bit Group Information," explains information needed to interpret the returned data string under Function 13 **Copy OIA**.

- Appendix D, "Related Publications," refers you to IBM books that you might need.
- Appendix E, "Sample Programs," contains code samples.
- Appendix F, "Glossary," defines acronyms and terms used in this manual.
- Appendix G, "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 EEHLLAPI.

A working knowledge of the Personal Computer and IBM Personal Computer DOS is assumed. If you want more information about the Personal Computer, refer to the list of IBM Personal Computer publications in Appendix D, "Related Publications."

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 EEHLLAPI, you must understand the following hardware, software, and host computer requirements.

Hardware Requirements

EEHLLAPI is designed to operate on the IBM Personal Computer, the IBM Personal Computer AT, the IBM Personal Computer XT, the IBM Personal Computer XT Model 286, and IBM Personal System/2™ Models 25, 30, 50, 60, and 80. It requires approximately 12,000 bytes of storage in addition to the requirements for DOS and the Entry Level Emulation Program.

Software Requirements

The following software guidelines should help you use EEHLLAPI:

- DOS Levels 3.2 and 3.3 are supported.
- Level 1.21 of the Entry Level Emulation Program.
- You must write your EEHLLAPI program in one of the following high-level languages (or its equivalent):
  - Compiled BASIC 1.0 or 2.0
  - Interpretive BASIC 2.1, 3.1, or 3.2
  - IBM COBOL 1.0
  - IBM C 1.0
  - IBM PASCAL 2.0.

™ IBM Personal System/2 is a trademark of the International Business Machines Corporation.

- Macro Assembler.

- Host graphics are not supported by EEHLLAPI.

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 Entry Level Emulation Program 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 3270 PC. This compatibility provides for easy migration of programs from the IBM PC 3270 Entry Level Emulation Program to the 3270 PC. Refer to the latest level of the *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 Entry Level Emulation Program.

Contents

EEHLLAPI Functions by Function Number

Chapter 1. Introduction

EEHLLAPI Overview

The Entry Emulator High-Level Language Application Program Interface or EEHLLAPI (pronounced E-E hil-lappy) lets you write and use personal computer programs in IBM BASIC, PASCAL, COBOL, C, or Macro Assembler to interact with your host session.

EEHLLAPI is valuable in a variety of work environments. It can increase productivity for experienced users and provide a shorter learning curve for inexperienced users.

1. It improves overall ease of use by:
   - Automating repetitive tasks
   - Masking complex applications from the user
   - Consolidating several complicated tasks into one simple task.
2. It simplifies existing host applications.

3. It provides unattended operation, a **programmed operator** that monitors tasks without human intervention. This programmed operator could:

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

4. You can 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. You can write programs that divide the work between host and PC sessions. EEHLLAPI 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, etc.) can be done in the PC session and 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. The goal is simply to finish the task.

In other words, you can use EEHLLAPI with the functions provided by the Entry Level Emulation Program to enhance interaction between an IBM Personal Computer application program and a host session **and** to simplify your programming tasks.

What You Need to Get Started

To use EEHLLAPI, you'll need the following:

1. The first thing you will need is an application program that you or someone else has written using IBM BASIC, COBOL, PASCAL, C, or Macro Assembler to call on the EEHLLAPI functions. If you don't have one yet, you can try out the Program Sampler that is provided on the Entry Emulator diskette (see Chapter 2 for more information).

   The other files that you'll need are on your Entry Emulator diskette.

2. `PC3270.COM` - this is the base Entry Emulator code.

3. `EEHLLAPI.EXE` - this is the EEHLLAPI interface module. It lets your program interact with the base code.

4. Except for interpretive BASIC, and assembly 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 on your IBM PC. It must be loaded before loading EEHLLAPI.

The Resident Module (EEHLLAPI.EXE)

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

Language Interface Modules (LIMs)

LIMs are the "bridges" between your EEHLLAPI application program and EEHLLAPI.EXE. LIMs are provided because each implementation of a programming language has its own unique methods of storing data and calling subroutines. LIMS allow EEHLLAPI 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 EEHLLAPI is loaded. Refer to Chapter 4, "Compiling and Running Your EEHLLAPI Application Program," for details.

Programs written using the MACRO Assembler can invoke EEHLLAPI 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 EEHLLAPI Program with Its 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. Refer to Chapter 4, "Compiling and Running Your EEHLLAPI Application Program" for details.

Function Calls

EEHLLAPI is a "function-code"—driven system. This means that each EEHLLAPI 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.

Notes

Chapter 2. Loading EEHLLAPI and Running the Program Sampler

This chapter tells you:

- How to load EEHLLAPI
- How to load and run the Program Sampler
- How to load EEHLAPI automatically.

Loading EEHLLAPI

To load EEHLLAPI:

1. Load DOS.
2. Place the Entry Level Emulation Program diskette in your default diskette drive.
3. Type `PC3270 r` next to the DOS prompt and press Enter (↵) to load the base code.

   Note: You must load the entry emulator in resume mode (r option when initially loading the PC3270 program, or type Alt-R while at the host when the PC3270 program is already loaded) before loading EEHLLAPI.

4. Next, type `eehllapi` and press Enter (↵) to load the EEHLLAPI code.

   Note: The base code must always be loaded first.

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

```
Entry Emulator High Level Language Application Program Interface 1.1
(c) Copyright International Business Machines Corporation 1984,1987

EHL001 EEHLLAPI is loaded

EHL002 EEHLLAPI is ready for use

A>
```

If you receive any other message refer to Appendix A, "EEHLLAPI Messages."

Note: If you want to load EEHLLAPI everytime you start your PC, see "Loading EEHLLAPI Automatically" on page 2-4.

Installing and Running the Program Sampler

After you have successfully loaded EEHLLAPI, you are ready to use the EEHLLAPI interface. The EEHLLAPI diskette contains a Complied BASIC Program Sampler (EHLSAMP.EXE) 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 EEHLLAPI functions have been used. When you get ready to write your EEHLLAPI application program, you may find it useful to refer to the BASIC sample program in Appendix E, "Sample Programs" on page E-1.

To run the Program Sampler EHLSAMP.EXE:

1. Have your Entry Level Emulation Program and EEHLLAPI.EXE loaded into storage.

2. Be sure that the following files are available to the program sampler (on the same diskette or in the same fixed disk directory):

   PC utilities that EHLSAMP uses, such as

   - Send.com
   - Receive.com.

3. Place your Entry Level Emulator Program diskette which contains EHLSAMP.EXE in Drive A.

4. Type **ehlsamp** and press Enter. You will see the EEHLLAPI Program Sampler panel appear.

5. Sample the EEHLLAPI functions.

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 EEHLLAPI 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 Function 1 **Connect**) before you can send keys to it (using Function 3 **Send Key**).

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 main menu.

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

Loading EEHLLAPI Automatically

If you load your Entry Level Emulation Program using an AUTOEXEC.BAT file, you can also load EEHLLAPI automatically by adding a line to your AUTOEXEC.BAT file. Edit the AUTOEXEC.BAT file, using any editor that works on the personal computer. If you don't have a standard personal computer editor, you can use the editor that comes with DOS (EDLIN). See your DOS manual for more details.

To load EEHLLAPI using an AUTOEXEC.BAT file, add "eehllapi" following the "PC3270" statement, as follows:

```
pc3270 r
eehllapi
```

Preparing a Diskette

If you want to use EEHLLAPI on an everyday basis, you will need to copy over certain files to your system diskette. The **Install.bat** file that is on the original diskette contains a list of all the files in the diskette along with comments that explain why the files are needed. To display this file on your screen, enter:

```
type Install.bat
```

To type this file to your PC printer, enter:

```
type Install.bat>prn
```

Copy the Install.bat file and use a PC editor (e.g., Personal Editor, EDLIN) to modify it to copy the files you need.

Notes

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
- Parameters Required When Called
- Values Returned
- Notes on Using This Function.

Parameters Required When Called

"Parameters Required When Called" 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 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 Emulation Program. When the PS position is not applicable, this 2-byte integer may be any value.

Values Returned

"Values Returned" 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 1 through 1920).

Return Code, the last parameter, is usually a numeric return code. Function 99 **Convert Position or RowCol** 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 and application development tips.

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

Connect Presentation Space (1)

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

Parameters Required When Called

Data String:	One-character short name of the host presentation space.*
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 Emulation Program, the default presentation space is "E".

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

Values Returned

The following return codes are valid:

Return Code	Explanation
0	**Connect** was successful; the host presentation space is unlocked and ready for input.
1	An invalid 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** call before using:

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

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

Parameters Required When Called

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 specify "P" above).

Values Returned

The 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 which is explained below:

Status Code	Explanation
0	Incorrect column or presentation space position was provided.
> 0	This is the PS position or column.
9998	An invalid 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 in the host-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 Function 31 **Find Field Position** and Function 32 **Find Field Length**. 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 conditions 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.

Parameters Required When Called

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.

Values Returned

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

Return Code	Explanation
0	**Copy Field to String** 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 may have been truncated because the string length may have been smaller than the field copied.
7	The host presentation space position is invalid.
9	A system error was encountered.

Copy OIA (13)

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

Parameters Required When Called

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.

Values Returned

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.

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.

Copy Presentation Space 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 Function 9 **Set Session Parameters**.)

Parameters Required When Called

Data String:	Preallocated target area the size of your host presentation space. This is 1920 for Entry Level Emulation Program.
Length:	NA (length of the presentation space is implied)
PS Position:	NA

Values Returned

Return Code	Explanation
0	The host presentation space contents were copied to 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.

Notes on Using This Function

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

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 Emulation Program, 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.

Copy Presentation Space to String 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 Function 9 **Set Session Parameters.**

Parameters Required When Called

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 Emulation Program.

Values Returned

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 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	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 invalid.
9	A system error was encountered.

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 Function 9 **Set Session Parameters**). See "Set Session Parameters (9)" on page 3-65.
- 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.

Parameters Required When Called

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.

Values Returned

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

Copy String to Presentation Space (15)

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

Parameters Required When Called

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 Emulation Program.

Values Returned

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

Notes on Using This Function

- If you want to place the string data at a specific cursor location, use Function 7 **Query Cursor Location** first to get the PS Position of the cursor. Then place this value in the "PS Position" calling parameter.

- The string ends when an EOT is encountered in the string (if EOT mode was selected using Function 9 **Set Session Parameters**). See "Set Session Parameters (9)" on page 3-65.

- Even though Function 3 **Send Key** seems to accomplish the same purpose as **Copy String to Presentation Space**, 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. **Copy String to Presentation Space** provides a much faster input path to the host.

- The keyboard mnemonics (see Function 3 **Send Key**) cannot be sent using **Copy String to Field**.

- The source data (the string being copied) can be no larger that 1920 characters.

Disconnect Presentation Space (2)

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

Parameters Required When Called

Data String: NA
Length: NA
PS Position: NA

Values Returned

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

Notes on Using This Function

- 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).
- Your programmed 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.**

- **Disconnect Presentation Space** does not reset the session parameters to the defaults. Your programmed operator must call Function 21 **Reset System** to accomplish this. For example, a keyboard locked using the **Reserve** function will remain locked after the application has been disconnected.

- 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).

Find Field Length (32)

Find Field Length 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 host 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.

Parameters Required When Called

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

Values Returned

Length:
= 0 means PC or unformatted host presentation space.
> 0 means length of requested field in host presentation space.

Return Code: The following are valid:

Return Code	Explanation
0	**Find Field Length** 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 invalid.
9	A system error was encountered.
24	No such field was found.

Find Field Position (31)

Find Field Position 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.*

Parameters Required When Called

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

Values Returned

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 are valid:

Return Code	Explanation
0	**Find Field Position** 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 invalid.
9	A system error was encountered.
24	No such field was found.

Pause (18)

Pause 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** may be ended by a host event if a prior Function 23 **Start Host Notification** had been called.

Parameters Required When Called

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)
- Delay for more than a moderate length of time (20 minutes) before checking the system "time of day" clock and proceeding with your EEHLLAPI program execution
- 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.

Values Returned

Return Code	Definition
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 Function 24 **Query Host Update** 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	Explanation
FPAUSE	Full-duration pause.
IPAUSE	Interruptible pause. Function 23 **Start Host Notification** and a host event will satisfy a Pause.

Once a pause has been satisfied by a host event, you *must* call Function 24 **Query Host Update** 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 Function 24 **Query Host Update** 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 **Query Cursor Location**, you must be connected to the host presentation space.

Parameters Required When Called

Data String: NA
Length: NA
PS Position: NA

Values Returned

Length: Presentation space position of the cursor.

Return Code: One from the table below:

Return Code	Explanation
0	**Query Cursor Location** 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.

Parameters Required When Called

Data String:	NA
Length:	NA
PS Position:	Presentation space position in the connected presentation space

Values Returned

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

Return Code: The following are valid:

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

Query Host Update (24)

The **Query Host Update** function lets the programmed 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 don't need to be connected to the host presentation space to check for updates.

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

Parameters Required When Called

Data String:	One-character short name of the host presentation space.
Length:	NA (1 is implied).
PS Position:	NA

Values Returned

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

Query Session Status (22)

Query Session Status provides session-specific information.

Parameters Required When Called

Data String:	Short name of the target presentation space plus 17 bytes for returned data
Length:	18 bytes
PS Position:	NA

Values Returned

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)

Byte	Definition
Position 11	For the Entry Emulator 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 Codes

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

Query Sessions (10)

Query Sessions, in the case of the Entry Emulator, 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.

If this function is to be used in 3270 PC applications as well as in Entry Emulation applications, you should know that this function will return one 12-byte descriptor for each session type available (excluding WS CTRL). This means a potential 144 bytes could be returned (12 sessions X 12 bytes) in a 3270 PC environment.

Parameters Required When Called

Data String:	Preallocated string of 12 bytes.
Length:	From 12 to 144 bytes.
PS Position:	NA

Values Returned

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
Positions 2-9	Long name of session. For Entry Level Emulation Program this is HOST1.
Position 10	Session type (H = host). For 3270 PC N = Notepad and P = PC.
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 the Entry Level Emulation Program.

Return Code: The following are valid:

Return Code	Explanation
0	**Query Sessions** 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 **Query System** to determine the level of the Entry Level Emulation Program support and other system-related 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."

Parameters Required When Called

Data String:	Preallocated string of 35 bytes.
Length:	NA (Implied length is 35).
PS Position:	NA

Values Returned

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)

Byte	Definition
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.
Position 16	Reserved
Positions 17-18	NLS table code for 3270 PC. Reserved for Entry Emulator.
Position 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 which indicated the system error and contained the bad internal code.

Return Codes

Return Code	Explanation
0	**Query System** 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 Function 20 **Query System**. 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 Function 20.

Receive File (91)

Receive File is used to receive a file from the host session to the PC session. It is used the same way as the "receive" command is used in the Entry Level Emulation Program. However, the **Receive File** function may be called by an EEHLLAPI application program.

Parameters Required When Called

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 fixed disk, the file must be in the current subdirectory.

Values Returned

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 may be returned. For example, if you are using the Entry Level Emulation Program, file transfer message "TRANS003 File transfer complete" would send back a return code of 3. For a complete list of the file transfer messages, refer to Appendix A in the *IBM PC 3270 Emulation Program, Entry Level, User's Guide.*

   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 **Receive 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**) the 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 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 may be overlaid and may 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 Function 91 **RECEIVE File**. This is not necessarily the root directory or the directory in use when EEHLLAPI was loaded, all of which may 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 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.

Character	Value (in minutes)	Character	Value (in minutes)
1	.5	8	4.0
2	1.0	9	4.5
3	1.5	J	5.0
4	2.0	K	5.5
5	2.5	L	6.0
6	3.0	M	6.5
7	3.5	N	7.0

Parameter	Explanation
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 may 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 may 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)

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

Parameters Required When Called

Data String: NA
Length: NA
PS Position: NA

Values Returned

Return Code	Explanation
0	**Release** 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 Function 11 **Reserve**) before ending your EEHLLAPI application, you will be locked out of that session until another application releases it or until you call Function 21 **Reset System**.

Reserve (11)

Reserve 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 Function 12 **Release** or Function 21 **Reset System.**

Parameters Required When Called

Data String: NA
Length: NA
PS Position: NA

Values Returned

Return Code	Explanation
0	**Reserve** was successful.
1	Your EEHLLAPI program is not connected to the 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 may 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.

Parameters Required When Called

Data String: NA
Length: NA
PS Position: NA

Values Returned

Return Code	Definition
0	**Reset System** was successful.
9	A system error was encountered.

Search Field (30)

Search Field 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.

Parameters Required When Called

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 Function 9, **Set Session Parameters**.

Values Returned

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	**Search Field** was successful.
1	Your program has not issued a CONNECT to the emulated host session.
7	The host presentation space position is invalid.
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 Function 9 **Set Session Parameters** related directly to search operations:

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

Parameter	Explanation
SRCHFRWD	**Search** will be performed in an ascending direction.
SRCHBKWD	**Search** 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 Function 9 **Set Session Parameters** 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.

Search Presentation Space (6)

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

Parameters Required When Called

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 Function 9, **Set Session Parameters**.

Values Returned

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	**Search Presentation Space** 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 **Set Session Parameters**, but not found in the data string).
7	The host presentation space position is invalid.
9	A system error was encountered.
24	The search string was not found.

Notes on Using This Function

- **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 the Entry Level Emulation Program.
- The **Search Presentation Space** function is useful in determining when the host presentation space is available. If your programmed operator is expecting a specific prompt or message before sending data, **Search Presentation Space** allows you to check for the prompt message before

continuing. If the expected prompt has not yet been sent, your program can call Function 18 **Pause** or Function 24 **Query Host Update** and continue to call **Search Presentation Space** until a zero return code is received.

- Two sets of parameters under Function 9 **Set Session Parameters** relate directly to Search functions:

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

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

Search Presentation Space 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 may 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)

Send File 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 the Entry Level Emulation Program. However, the **Send File** function may be called by an EEHLLAPI application program.

Parameters Required When Called

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 fixed disk, the file must be in the current subdirectory.

Values Returned

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 Emulation Program file transfer messages may be returned. For example, if you are using the Entry Level Emulation Program, file transfer message "TRANS003 File transfer complete" would send back a return code of 3. For a complete list of the file transfer messages, refer to Appendix A in the *IBM PC 3270 Emulation Program, Entry Level, User's Guide.*

   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	Invalid function number
302	File not found
305	Access denied
308	Insufficient memory
310	Invalid environment
311	Invalid 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 subprocess applications are running, COMMAND.COM may be overlaid and may 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 may be changed by the time **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. Character / Value (in minutes) / Character / Value (in minutes) 1 .5 8 4.0 2 1.0 9 4.5 3 1.5 J 5.0 4 2.0 K 5.5 5 2.5 L 6.0 6 3.0 M 6.5 7 3.5 N 7.0
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.

For additional information, see "Considerations for Using Functions 90 and 91" on page 3-44.

Send Key (3)

Send Key sends a keystroke or a string of keystrokes to the host presentation space. Your EEHLLAPI application program must use Function 1 **Connect Presentation Space** 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, etc. All the fields that are protected for input or are numeric only must be treated accordingly.

Parameters Required When Called

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

Values Returned

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.

Notes on Using This Function

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

- Keystrokes cannot be sent to a session whose keyboard is locked; for example, when input is inhibited. You can check this with Function 4, **Wait**.
- Keystroke input is no longer accepted after the first AID character is received. The remainder of the input data string may 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 comprises 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 3-63.

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 Function 9 **Set Session Parameters** 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 Function 33 **Copy String to Field** or Function 15 **Copy String to Presentation Space** 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", etc. Please note that the upper and lower case alphabetic characters are mnemonic abbreviations for different keys.

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

@0 Home	@7 PF7	@e PF14	@1 PF21
@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	

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

AID Key Mnemonics

The AID key mnemonics set by Function 3 **Send Key** 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	@1	PF21		

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

@C - Clear
@E - Enter

You can specify an escape character other than "@" using an EOT delimiter defined using Function 9 **Set Session Parameters** 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 Function 9 **Set Session Parameters.**

Set Session Parameters (9)

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

Parameters Required When Called

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 Functions 90 **(Send File)** and 91 **(Receive File)**:

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 equals sign.

The next two sets of parameters affect Function 3 **Send Key**:

Parameter	Explanation
ESC=n	Specify the escape character for keystroke mnemonics (@ is the default). Do not leave a blank after the equals 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 Function 3 **Send Key** with a reset.
NORESET	Do not AUTORESET.

The next two sets of parameters affect Search functions:

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

Parameter	Explanation
SRCHFRWD	**Search** will be performed in an ascending direction.
SRCHBKWD	**Search** 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 4-16 for more information about this parameter.

Parameter	Explanation
TRON	Turns trace on.
TROFF	Turns trace off. The trace function may 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 Function 4 **Wait:**

Parameter	Explanation
TWAIT	**Wait** will wait up to a minute before timing out on XCLOCK or XSYSTEM.
LWAIT	**Wait** 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	**Wait** checks status and returns immediately (no wait).

Two parameters under Function 9 affect Disconnect:

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 Function 18 **Pause**:

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

The two next sets of parameters affect Function 90 **Send** and Function 91 **Receive**:

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 4.0 2 1.0 9 4.5 3 1.5 J 5.0 4 2.0 K 5.5 5 2.5 L 6.0 6 3.0 M 6.5 7 3.5 N 7.0
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.

Values Returned

Return Code	Explanation
0	The session parameters have been set.
2	The length of the parameter list is invalid.
9	A system error was encountered.

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 Function 24 **Query Host Update** to determine what specific host event has occurred.

Parameters Required When Called

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: • 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 may use Function 17 **Storage Manager** Subfunction 1 **Get Storage** to obtain a buffer block of storage. **Get Storage** allows all languages to get additional storage and to free smaller blocks of storage as needed.

Function 17 **Storage Manager** 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 may 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 Function 17* **Storage Manager***), you* must *call Function 25* **Stop Host Notification** *before you exit*

EEHLLAPI. (You could also use Function 21 **Reset System** *to accomplish the same purpose).* **Failure to do so may cause subsequent programs to fail unpredictably with storage overlays.**

Values Returned

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

Notes on Using This Function

In order to use this function, **Storage Manager** can be utilized. A **Get Storage** call should be made. See "Storage Manager (17)" on page 3-74.

Stop Host Notification (25)

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

Parameters Required When Called

Data String:	One-character short name of the target presentation space ID.
Length:	NA (implied length of 1 byte).
PS Position:	NA

Values Returned

Return Code	Definition
0	**Stop Host Notification** was successful.
1	An invalid 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 3-74.

Storage Manager (17)

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

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 **Storage Manager** 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 may 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 Function 17* **Storage Manager***), you* must *call Function 25* **Stop Host Notification** *before you exit EEHLLAPI. (You could also use Function 21* **Reset System** *to accomplish the same purpose.)* **Failure to do so may 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

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

Parameters Required When Called

Data String:	NA (preallocated to 4 bytes).
Length:	Size (in bytes) of the requested storage area.
PS Position:	01 (for Get Storage).

Values Returned

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 **Start Host Notification**.
- The Length parameter contains the storage block ID.

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

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

Free Storage

Free Storage frees the block of storage allocated by Subfunction 1 **Get Storage.** This function should be used after to a **Stop Host Notification** request in order to free "queue" memory created by a Get Storage call on **Start Host Notification.**

Parameters Required When Called

Data String:	NA (preallocated to 4 bytes).
Length:	ID of the storage block to be freed.
PS Position:	02 (for Free Storage).

Values Returned

The subfunction returns these return codes:

Return Code	Explanation
0	The storage has been freed.
2	The storage block ID was invalid, or 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.

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

Free All Storage 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.

Parameters Required When Called

Data String:	NA (preallocated to 4 bytes)
Length:	NA
PS Position:	04 (for Free All Storage)

Values Returned

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	**Free All Storage** 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	**Storage Manager** is not available.

Wait (4)

Wait 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.

Parameters Required When Called

Data String: NA

Length: NA

PS Position: NA

Values Returned

The following return codes are valid:

Return Code	Definition
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 locked.
9	A system error was encountered.

Notes on Using This Function

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

Parameter	Explanation
TWAIT	**Wait** will wait up to a minute before timing out on XCLOCK or XSYSTEM.
LWAIT	**Wait** 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	**Wait** checks status and returns immediately (no wait).

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

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

Notes

Chapter 4. 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 requirement 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 5 **Copy Presentation Space** 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 don't understand the routine involved, don't worry; just copy the following program statements into your program's initialization routine:

```
100 DEF SEG = 0: APILOC=PEEK(&H1FE) + PEEK(&H1FF)
110 IF APILOC = 0 THEN PRINT "EEHLLAPI IS NOT AVAILABLE": END
120 DATA &H04B8, &HB301, &HCDBB, &HCA7F, &H0002
130 DIM APICALL% (4)
140 FOR I=0 to 4: READ APICALL%(I): NEXT I
150 BLIMSET = VARPTR (APICALL%(0))
160 DEF SEG: CALL BLIMSET (APICALL%(1))
170 BLIM=APICALL%(1): DEF SEG = APICALL%(0)
180 IF BLIM = &HB301 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 fourparameters. These parameters must occur in a fixed sequence and all four parameters must be in the call list, even if they aren't used. If a parameter isn't needed for a given call, you don't need to value it.

The call format is:

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

Where:

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

Please 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 "Parameters Required When Called" on page 3-1 and of the returned parameters under "Values Returned" on page 3-2.

- The function code, data length, and return code **must** be two-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, Function 5 **Copy Presentation Space** is not available, but Function 8 **Copy Presentation Space to**

String may 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 may 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:

```
1000  SDATA$=SPACE$(80) 'PREALLOCATE TO 80 BYTES
1010  FUNC%=08    'COPYSTRING FUNCTION
1020  DLEN%=80    'COPY 80 BYTES
1030  RETC%=1     'LOCATIONS 1-80 FOR COPY
1035  REM NOTE USE OF RETC% TO PASS OFFSET FOR COPYSTRING
1040  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 may 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. In order 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 Emulator 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 Emulator 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-0.
77  RETCODE          PIC 99          COMP-0.
77  STRLEN           PIC 99          COMP-0.
01  DATA-STRING      PIC X(1920)     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 Emulator 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 your Entry Emulator 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 may 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. The 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 substringing 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 Function 20 **Query System** can be done as shown on the following page.

```
PROGRAM QSYS(INPUT,OUTPUT);

TYPE
                                    (* Query System overlay type     *)
                                    (* [xx] necessary for proper     *)
   QSYSOVLTYP = RECORD              (*  byte alignment               *)
    HLLVERS [00]: STRING(1);        (* EEHLLAPI version number       *)
    HLLLVL  [01]: STRING(2);        (* EEHLLAPI level number         *)
    HLLDATE [03]: STRING(6);        (* EEHLLAPI date                 *)
    LIMVERS [09]: STRING(1);        (* LIM version number            *)
    LIMLVL  [10]: 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 session short 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 return code             *)
 Q_SYS_OVL: QSYSOVLTYP;
                                (* 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 20)                          *)
  (* ------------------------------------------------------------ *)

  FUNC := 20;
  LEN  := 35;                               (* Data String Length   *)
  RETC := 0;

  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 Emulator 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 Emulator 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 may 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. The 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 IBM 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 Compile, 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 Emulator 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 may 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 detail 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 available:

1. Have your Entry Level Emulation Program 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 Emulation 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 Function 9 **Set Session Parameters**. 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 return code was returned.

The Trace function may 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."

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:

- EHLnnn 'Message Text'

Where 'nnn' is a message ID.

The messages are listed below in numerical 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** 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**, 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 Emulation 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 Emulation 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 service coordinator.

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 may 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 may overlap with messages from programs that track their own screen management (i.e., BASIC). The value is displayed in hexadecimal notation.

Notes

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 may 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 may wish 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 B-1 on page B-3 as you read this information.

Writing Your Own LIM

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 Emulation Program.

5. The Entry Level Emulation 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 looks something like this:

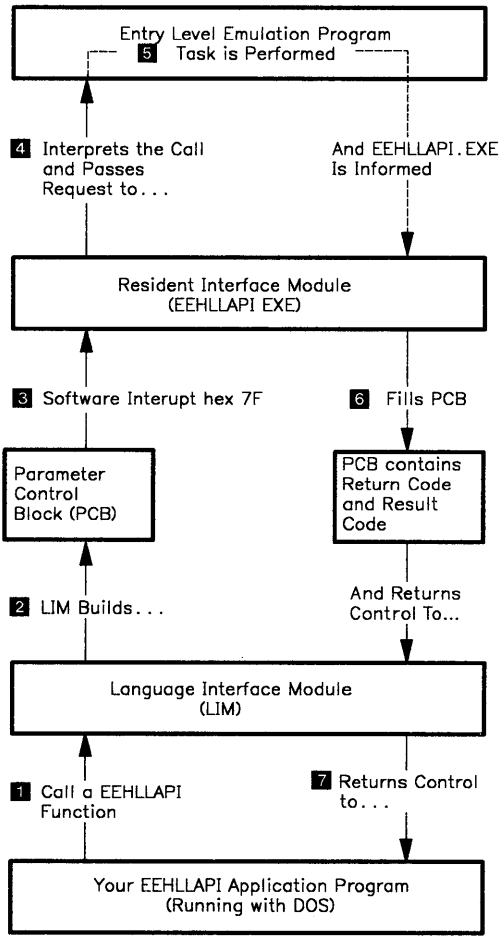

Figure B-1. The Parts of EEHLLAPI

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.
- Receive control back from EEHLLAPI.EXE.
- Pass back 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 may 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 may 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 IBM-supplied LIMs are designed to provide a fixed-format, easy-to-use call format. You may wish 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 may want to redesign the function calls to use a row/column format.

In that 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 the call 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 a 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    0       ;Function code VALUE
PCB_DSEG     DW    0       ;String seg addr
PCB_DADDR    DW    0       ;String offset addr
PCB_LENGTH   DW    0       ;Data length VALUE
PCB_FILLER   DB    0       ;Unused
PCB_RETCODE  DW    0       ;Return code VALUE
PCB_STRLEN   DW    25000   ;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 - 0104H (mandatory)
           BX - rrAAH (BL must = AAhex),
                      rr=01-04 request code
                      (01=get, 02=free, 04=freeall)
           CX - storage size (on GET request)
                block ID (on FREE request)

Returns:
Call:     01 (get):    CX=block ID, DX=RETC
                                    AX=storage seg
                                    BX=offset
          02 (free):                DX=RETC
          04 (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. IBM-supplied LIMs support this call. User-supplied LIMs may or may 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 non-segment 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 and result codes. This allows you to examine and, if necessary, reformat these values.

Notes

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 Function 13 **Copy OIA**.

OIA Image Group (Positions 2 through 81)

The meaning of symbols in the OIA Image Group are understood by looking up the desired hexadecimal code in Figure C-1 "Host Presentation Space Character Table" on page C-2. This figure shows the hexadecimal codes found in the host presentation space, and the characters they represent. Refer to the *IBM PC 3270 Emulation Program, Entry Level, User's Guide* for information on the OIA indicators.

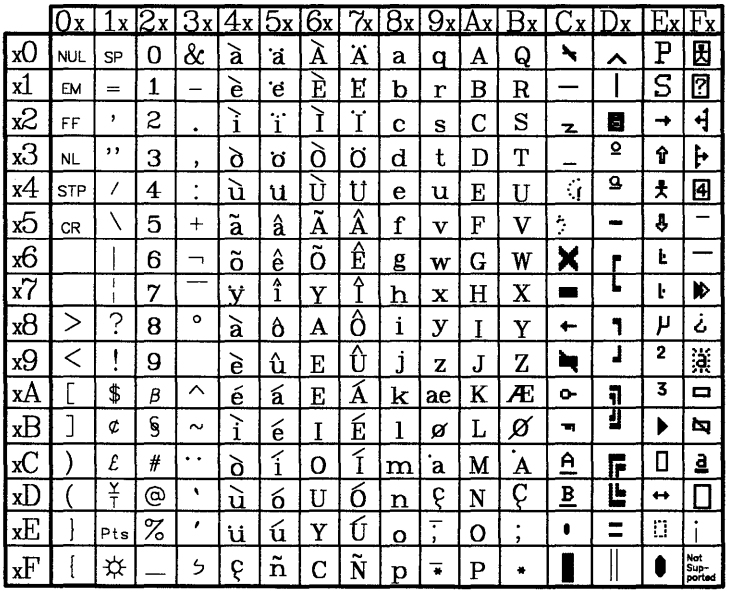

Figure C-1. Host Presentation Space Character Table

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 Emulation Program. 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

Notes

Appendix D. Related Publications

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

IBM PC Disk Operating System

IBM PC 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

IBM PC 3270 Emulation Program, Entry Level, User's Guide

IBM 3270 PC High Level Language Application Program Interface, Programming Guide.

Notes

Appendix E. 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

```
10 REM EEHLLAPI Version 1.XX
20 REM SAMPLE PROGRAM
30 REM (c) COPYRIGHT 1986,1987, IBM CORPORATION
40 KEY OFF: CLS
50 LOCATE 1,10 : PRINT "      EEHLLAPI SAMPLE BASIC PROGRAM"
60 LOCATE 2,10 : PRINT "(c)Copyright 1986,1987 IBM Corporation"
70 CLEAR
80 REM This sample basic program uses EEHLLAPI to provide an
90 REM interface to the user for downloading multiple files
100 REM from the host.
110 REM ------------------------------------------------------
120 REM *** MANDATORY INT BASIC INITIALIZATION ****
130 REM ------------------------------------------------------
140 DEF SEG = 0: APILOC=PEEK(&H1FE) + PEEK(&H1FF)
150 IF APILOC = 0 THEN PRINT "EEHLLAPI IS NOT AVAILABLE": END
160 DATA &H04B8, &HB301, &HCDBB, &HCA7F, &H0002
170 DIM APICALL% (4)
180 FOR I=0 TO 4: READ APICALL%(I): NEXT I
```

```
190 BLIMSET = VARPTR (APICALL%(0))
200 DEF SEG: CALL BLIMSET (APICALL%(1))
210 BLIM=APICALL%(1): DEF SEG = APICALL%(0)
220 IF BLIM = &HB301 THEN PRINT "EEHLLAPI NOT AVAILABLE": END
230 REM ------------------------------------------------------
240 REM *** END OF REQUIRED INIT. LOGIC ***
250 REM ------------------------------------------------------
260 GOTO 380
270 REM ------ Common routine to call EEHLLAPI ---------------
280 CALL BLIM (FUNC%,SDATA$,DLEN%,RETC%)
290 IF RETC% = 9 AND FUNC% <> 99 THEN GOTO 310
300 RETURN
310 REM --- System Error ---
320 CLS
330 PRINT "A System Error was detected during function:";FUNC%
340 FUNC%=20: SDATA$ = SPACE$(35)
350 CALL BLIM (FUNC%,SDATA$,DLEN%,RETC%)
360 PRINT "Error component information ="; MID$(SDATA$,20,8)
370 END
380 REM ======================================================
390 REM ---------------------- MAIN PROGRAM ------------------
400 REM ======================================================
410 REM ============ RESET SYSTEM =============
420 FUNC% = 21
430 GOSUB 270
440 REM ---------------------------------------
450 CLS
460 LOCATE  5,5 : PRINT "Drive # (1..26) for RECEIVE.COM";
470 INPUT " : " ,DRIVE%
480 HOST$ = "E"
490 REM >>> Connect to Host Session
500 REM =============== CONNECT ===============
510 FUNC% = 1
520 SDATA$ = HOST$
530 GOSUB 270
540 REM ---------------------------------------
550 IF RETC% <> 0 THEN PRINT "Could not connect to host" : END
560 REM >>> Clear Host Screen
570 REM =============== SENDKEY ===============
580 FUNC% = 3
590 SDATA$ = "@C"
600 DLEN% = 2
610 GOSUB 270
620 REM ---------------------------------------
630 CLS
640 REM --------- PROMPT FOR FILE SPECS ---------
650 LOCATE  5,1
660 PRINT "Please input the info for files to be ";
670 PRINT "considered for downloading from the host."
680 PRINT "Use the wild card character (*) in order ";
690 PRINT "to consider a group of files."
700 LOCATE 11,15: INPUT "filename : ",FILENAME$
710 LOCATE 13,15: INPUT "filetype : ",FILETYPE$
720 LOCATE 15,15: INPUT "filemode : ",FILEMODE$
730 FILELIST$ = FILENAME$ + " " + FILETYPE$ + " " + FILEMODE$
740 REM >>> Filelist Command
750 REM =============== SENDKEY ===============
760 FUNC% = 3
770 SDATA$ = "filelist " + FILELIST$ + "@E"
780 DLEN% = LEN(SDATA$)
```

```
790 GOSUB 270
800 REM ---------------------------------------
810 REM >>> Wait for Host to execute filelist command
820 REM ================ WAIT ================
830 FUNC% = 4
840 GOSUB 270
850 REM ---------------------------------------
860 REM >>> Determine Position of Header Field
870 REM ========= FIND FIELD POSITION =========
880 FUNC% = 31
890 SDATA$ = "N "
900 RETC% = 1
910 GOSUB 270
920 REM ---------------------------------------
930 IF RETC% = 0 THEN GOTO 950
940 PRINT "Could not find position of header field" : END
950 FPOS% = DLEN%
960 IF FPOS% < 200 THEN GOTO 980
970 PRINT "Found bad header field position" : END
980 REM >>> Determine Size of Header Field
990 REM ========== FIND FIELD LENGTH ==========
1000 FUNC% = 32
1010 SDATA$ = "T "
1020 RETC% = FPOS%
1030 GOSUB 270
1040 REM ---------------------------------------
1050 FLEN% = DLEN%
1060 REM >>> Search Header Field for 'Filename'
1070 REM ============== SRCH FIELD =============
1080 FUNC% = 30
1090 SDATA$ = "Filename"
1100 DLEN% = 8
1110 RETC% = FPOS%
1120 GOSUB 270
1130 REM ---------------------------------------
1140 IF RETC% <> 24 THEN GOTO 1160
1150 PRINT "Could not find header string" : END
1160 REM >>> The file info (name,type,mode) in
1170 REM >>> each field begins under 'Filename'
1180 OFFSET% = DLEN% - FPOS%
1190 CLS
1200 LOCATE 4,1
1210 PRINT "Please jump to your Host session and ";
1220 PRINT "put an asterisk (*) in the cmd field "
1230 PRINT "for each file that you would like ";
1240 PRINT "downloaded from the host.  The set "
1250 PRINT "of files to be downloaded must include ";
1260 PRINT "the first file and any number of "
1270 PRINT "consecutive files after the first. ";
1280 PRINT "Then jump back to your PC session and "
1290 INPUT "press enter to continue . . .",X$
1300 CLS
1310 REM --- ============================ ---
1320 REM --- BEGIN LOOP GET/TRANSFER FILE ---
1330 REM --- ============================ ---
1340 CLS
1350 REM >>> Find the beginning of the next field
1360 REM ========= FIND FIELD POSITION =========
1370 FUNC% = 31
1380 SDATA$ = "N "
```

```
1390 RETC% = FPOS%
1400 GOSUB 270
1410 REM ---------------------------------------
1420 IF RETC% = 0 THEN GOTO 1440
1430 PRINT "Could not find next field" : END
1440 FPOS% = DLEN%
1450 IF FPOS% > 1441 THEN PRINT "Bad Field" : END
1460 REM >>> Search cmd field for flag *
1470 REM ============== SRCH FIELD =============
1480 FUNC% = 30
1490 SDATA$ = "*"
1500 DLEN% = 1
1510 RETC% = FPOS%
1520 GOSUB 270
1530 REM ---------------------------------------
1540 REM >>> If no flag, then all done
1550 IF RETC% = 24 THEN GOTO 2150
1560 REM >>> Copy file info from field
1570 REM ========= COPY FIELD TO STRING ========
1580 FUNC% = 34
1590 DLEN% = FLEN%
1600 SDATA$ = SPACE$(DLEN%)
1610 RETC% = FPOS% + OFFSET%
1620 GOSUB 270
1630 REM ---------------------------------------
1640 HOSTFILE$ = MID$(SDATA$,OFFSET% + 1,20)
1650 REM --- remove spaces from file spec string ---
1660 TEMP% = INSTR(HOSTFILE$," ")
1670 IF TEMP% = 0 THEN GOTO 1720
1680 TEMP1$ = MID$(HOSTFILE$,1,TEMP%)
1690 TEMP2$ = MID$(HOSTFILE$,TEMP% + 2,LEN(HOSTFILE$))
1700 HOSTFILE$ = TEMP1$ + TEMP2$
1710 GOTO 1660
1720 REM --- Get PC File specs ---
1730 CLS
1740 LOCATE  5,5 : PRINT "DOWNLOADING :"
1750 LOCATE  7,25: PRINT HOSTFILE$
1760 LOCATE 10,5
1770 PRINT "Please input PC file specifications."
1780 LOCATE 11,5
1790 PRINT "If you specify a path, be ";
1800 PRINT "be sure that the path ends in -"
1810 LOCATE 13,15: INPUT "path     : ",PCPATH$
1820 LOCATE 15,15: INPUT "filename : ",PCNAME$
1830 LOCATE 17,15: INPUT "filetype : ",PCTYPE$
1840 PCFILE$ = PCPATH$ + PCNAME$ +  "." + PCTYPE$
1850 LOCATE 19,15: INPUT "options  : ", OPTIONS$
1860 OPTIONS$ = "(" + OPTIONS$
1870 REM >>> Must Disconnect in order to call Function 91
1880 REM ============= DISCONNECT ==============
1890 FUNC% = 2
1900 GOSUB 270
1910 REM ---------------------------------------
1920 CLS
1930 LOCATE 1,1
1940 PRINT "Downloading : ";HOSTFILE$;" to ";PCFILE$
1950 LOCATE 3,1
1960 REM ============= RECEIVE FILE ============
1970 FUNC% = 91
1980 SDATA$ = PCFILE$ + " " + HOSTFILE$ + " " + OPTIONS$
```

```
1990 DLEN% = LEN(SDATA$)
2000 RETC% = DRIVE%
2010 GOSUB 270
2020 REM ---------------------------------------
2030 LOCATE 24,10 : INPUT "Press Enter to continue....",X$
2040 REM >>> Reconnect to Host Session
2050 REM =============== CONNECT ===============
2060 FUNC% = 1
2070 SDATA$ = HOST$
2080 GOSUB 270
2090 REM ---------------------------------------
2100 GOTO 1310
2110 REM --- ========================== ---
2120 REM --- END LOOP GET/TRANSFER FILE ---
2130 REM --- ========================== ---
2140 REM >>> Get out of filelist screen
2150 REM =============== SENDKEY ===============
2160 FUNC% = 3
2170 SDATA$ = "@3"
2180 DLEN% = 2
2190 GOSUB 270
2200 REM ---------------------------------------
2210 REM >>> Reset System to known state (Disconnect, etc.)
2220 REM ============ RESET SYSTEM =============
2230 FUNC% = 21
2240 GOSUB 270
2250 REM ---------------------------------------
2260 CLS
2270 LOCATE 2,5 : PRINT "DOWNLOADING IS COMPLETED"
2280 LOCATE 5,5 : INPUT "(b) BASICA or (d) DOS ? ",SCONT$
2290 CLS
2300 IF SCONT$ = "b" THEN GOTO 2330
2310 IF SCONT$ = "d" THEN GOTO 2340
2320 GOTO 2340
2330 KEY ON : END
2340 SYSTEM
2350 ============================================================
```

COBOL Sample Program

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID. SAMPLER.
*   This is 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-0.
 77  DLEN                    PIC 99 COMP-0.
 77  RETC                    PIC 99 COMP-0.
 77  DATA-STR                PIC X(80) VALUE SPACES.
*----MISC INPUT STRINGS----*
 77  PSID-STR                PIC X VALUE SPACES.
 77  EMPTY-STR               PIC X(80) VALUE SPACES.
 01  HOST-EVENT-BUFF.
     03  HE-PSID             PIC X VALUE ' '.
     03  HE-TYPE             PIC X VALUE 'B'.
     03  HE-PTR              PIC X(4) VALUE LOW-VALUES.
     03  FILLER              PIC X(256).
*============================================================
 SCREEN SECTION.
 01  NUMER-MENU.
*----NUMERICAL LIST OF FUNCTIONS----*
     03 BLANK SCREEN.
     03 LINE  1 COLUMN 24 VALUE 'EEHLLAPI NUMERICAL MENU'.
     03 LINE  2 COLUMN 24 VALUE '======================'.
     03 LINE  3 COLUMN 1  VALUE ' 1 - Connect'.
     03 LINE  4 COLUMN 1  VALUE ' 2 - Disconnect'.
     03 LINE  5 COLUMN 1  VALUE ' 3 - Send Key'.
     03 LINE  6 COLUMN 1  VALUE ' 4 - Wait'.
     03 LINE  7 COLUMN 1  VALUE ' 6 - Search PS'.
     03 LINE  8 COLUMN 1  VALUE ' 7 - Q Cursor Location'.
     03 LINE  9 COLUMN 1  VALUE ' 8 - Copy PS to String'.
     03 LINE 10 COLUMN 1  VALUE ' 9 - Set Session Parms'.
     03 LINE 11 COLUMN 1  VALUE '10 - Q Sessions'.
     03 LINE  3 COLUMN 27 VALUE '11 - Reserve'.
     03 LINE  4 COLUMN 27 VALUE '12 - Release'.
     03 LINE  5 COLUMN 27 VALUE '13 - Copy OIA'.
     03 LINE  6 COLUMN 27 VALUE '14 - Q Field Attribute'.
     03 LINE  7 COLUMN 27 VALUE '15 - Copy String to PS'.
     03 LINE  8 COLUMN 27 VALUE '18 - Pause'.
     03 LINE  9 COLUMN 27 VALUE '20 - Q System'.
     03 LINE 10 COLUMN 27 VALUE '21 - Reset System'.
     03 LINE 11 COLUMN 27 VALUE '22 - Q Session Status'.
     03 LINE  3 COLUMN 53 VALUE '23 - Start Host Notify'.
     03 LINE  4 COLUMN 53 VALUE '24 - Q Host Update'.
     03 LINE  5 COLUMN 53 VALUE '25 - Stop Host Notify'.
     03 LINE  6 COLUMN 53 VALUE '30 - Search Field'.
     03 LINE  7 COLUMN 53 VALUE '31 - Find Field Pos.'.
     03 LINE  8 COLUMN 53 VALUE '32 - Find Field Length'.
     03 LINE  9 COLUMN 53
        VALUE '33 - Copy String to Field'.
     03 LINE 10 COLUMN 53
        VALUE '34 - Copy Field to String'.
     03 LINE 11 COLUMN 53 VALUE '99 - Convert Pos/RowCol'.
 01  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.
      03  LINE 14 COLUMN 20 VALUE 'Select a function: '.
      03  LINE 14 COLUMN 39 PIC ZZ9 TO CHOICE.
      03  LINE 23 COLUMN  1 BLANK LINE.
      03  LINE 24 COLUMN 20 VALUE '300-Exit Sample Program'.
*----PROMPT SCREENS-----------------------------------------
 01  STRING-PROMPT.
*    This screen is used to prompt the user for the
*    Data String calling parameter.
      03  LINE 15 COLUMN 13 VALUE 'String : '.
      03  LINE 15 COLUMN 23 PIC X(55) TO DATA-STR.
 01  PSID-PROMPT.
*    This screen is used to prompt the user for the
*    Data String calling parameter when the Data String
*    only needs a PSID.
      03  LINE 15 COLUMN 13 VALUE 'PSID : '.
      03  LINE 15 COLUMN 21 PIC X TO PSID-STR.
 01  HOST-STR-PROMPT.
*    This screen is used to prompt the user for the
*    PSID and option code for function 23, Start Host
*    Notification.
      03  LINE 15 COLUMN 13 VALUE 'PSID : '.
      03  LINE 15 COLUMN 21 PIC X TO HE-PSID.
      03  LINE 15 COLUMN 25 VALUE 'option code : '.
      03  LINE 15 COLUMN 39 PIC X TO HE-TYPE.
 01  LENGTH-PROMPT.
*    This screen is used to prompt the user for the
*    Length calling parameter.
      03  LINE 16 COLUMN 13 VALUE 'Length : '.
      03  LINE 16 COLUMN 23 PIC ZZZ9 TO DLEN JUST.
 01  RC-PROMPT.
*    This screen is used to prompt the user for the
*    PS Position/Return Code calling parameter.
      03  LINE 17 COLUMN 13 VALUE 'PS Position : '.
      03  LINE 17 COLUMN 28 PIC ZZZ9 TO RETC JUST.
*----DISPLAY SCREENS----------------------------------------
 01  STRING-DISPLAY.
*    This screen displays the returned Data String.
      03  LINE 19 COLUMN 13 VALUE 'String : '.
      03  LINE 19 COLUMN 23 PIC X(55) FROM DATA-STR.
 01  LENGTH-DISPLAY.
*    This screen displays the returned Length.
      03  LINE 20 COLUMN 13 VALUE 'Length : '.
      03  LINE 20 COLUMN 23 PIC ZZZ9 FROM DLEN-DISPLAY.
 01  RC-DISPLAY.
*    This screen displays the returned Return Code.
      03  LINE 21 COLUMN 13 VALUE 'Return Code : '.
      03  LINE 21 COLUMN 28 PIC ZZZ9 FROM RETC-DISPLAY.
*----MISCELLANEOUS SCREENS----------------------------------
 01  BLANK-SCR.
      03 BLANK SCREEN.
 01  BLANK-LINE.
      03 BLANK LINE.
 01  CONT-SCR.
      03  LINE 23 COLUMN 23 BELL REVERSE-VIDEO
          VALUE 'Press Enter to continue'.
      03  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 0 TO CHOICE.
     DISPLAY CHOICE-SCR.
     ACCEPT CHOICE-SCR.
*   Does the user want to exit the program?
     IF CHOICE = 300 GO TO DONE.
*----INITIALIZE VARIABLES-------------------------------
     MOVE CHOICE TO FUNC.
     MOVE 0 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 = 10 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-ATTRIB.
     IF FUNC = 15 PERFORM COPY-STRING-PS.
     IF FUNC = 18 PERFORM PAUSE.
     IF FUNC = 20 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 = 30 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-LENGTH-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
Send Key	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 information 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 := 20;
      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 := 20 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 := 10;
   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 10th position *)
      if Data_String[offset+10] = '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('0') ;
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 := 10 * digit(time_string[1]) +
                     digit(time_string[2]) ;
   time.min := 10 * digit(time_string[4]) +
                      digit(time_string[5]) ;
   time.sec := 10 * digit(time_string[7]) +
                      digit(time_string[8]) ;
   time.tick := 0;
   date.month := 10 * digit(date_string[1]) +
                      digit(date_string[2]) ;
   date.day := 10 * digit(date_string[4]) +
                     digit(date_string[5]) ;
   date.year := 10 * digit(date_string[7]) +
                     digit(date_string[8]) +
                     1900;
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 > 0 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;
   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 > 0)     /* 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 >= 0; --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\0");
    API_LEN = 1;
    hllc(&API_FUNC,API_STRING,&API_LEN,&API_RETC);
    if (API_RETC != 0)
       {
        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 == 0)
       {
        cursor_pos = API_LEN;
        printf("Cursor 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\0");
    API_RETC = cursor_pos;
```

```
hllc(&API_FUNC,API_STRING,&API_LEN,&API_RETC);
if (API_RETC == 0 | | API_LEN == 0)
   {
    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 == 0)
   {
    printf("The screen is unformatted.\n");
    return;
   }
else if (API_RETC != 0)
   {
    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 = 0; 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.\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 == 0 )
          {
           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!='/012') getchar();
    if (ch == 'i') get_info();
   }
  while (ch != 'q');

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

}
```

Appendix F. Glossary

AID key. A control key that generates a host attention interrupt.

American National Standard Code for Information Interchange (ASCII). One of the two standard codes used for exchanging information among data processing systems and associated equipment; the standard code used by the IBM Personal Computer and other microcomputers.

attention identifier. See "AID key."

attribute byte. In 3270 application, the byte used for determining the characteristics of the following field (color, protected, highlighted).

autoskip. A field defined as protected **and** numeric. Causes the cursor to skip to the next unprotected field.

BASIC. Beginner's All-Purpose Symbolic Instruction Code. A high-level, widely used computer programming language.

EEHLLAPI.EXE. See "resident interface module."

extended error code. An eight-byte data string returned by **Query System** generated by an internal system error that is used by service personnel for diagnosis.

field. A group of consecutive positions on a presentation space with similar characteristics. These characteristics are defined by the field attribute byte at the beginning of the field.

field-formatted presentation space. A presentation space which is made up of one or more fields.

HLLAPI. High-Level Language Application Program Interface.

language interface module (LIM). Programs provided by EEHLLAPI or that you can write that serve as bridges between your program and the resident interface module (EEHLLAPI.EXE) that passes parameters to the Entry Level Emulation Program.

logging on. The procedure by which you are linked to a multiple-user host computer system; the procedure requires a user identification and generally a password.

menu. A list of available operations. You select which operation you want from the list.

NA. Not applicable. When this appears in a calling parameter position, it means that EEHLLAPI will not read the data in these fields.

operator information area (OIA). The bottommost line of your screen where you receive information about the status of your host.

presentation space (PS). An area in storage that corresponds to the image on your screen.

presentation space position parameter. One of the four parameters that you must specify for each EEHLLAPI function. A position in the presentation space from 1 to 1920.

programmed operator. The software "user" that performs and monitors activities in your PC without actual human intervention; your EEHLLAPI application program.

protected field. A field that is protected from modification by the operator.

PS. See "presentation space."

PSID. Presentation Space Identifier; short one-character or one-letter name of the presentation space.

resident interface module (EEHLLAPI.EXE). The main interface module for EEHLLAPI; it must be loaded before you can use EEHLLAPI and must remain in storage as a resident extension of DOS.

service coordinator. The person in your organization responsible for answering hardware and software computing questions.

session. A connection between your work station and a host computer. (Contrast with "presentation space.")

short name. The one-letter name (A through Z) of the host presentation space. For the Entry Level Emulation Program this is always "E".

terminal operator. The human user of an EEHLLAPI application program (contrast with "programmed operator").

unprotected field. A field that is available for the operator to enter or modify data.

valid key. A key that is recognized by the host session.

Notes

Appendix G. Alternate Code Page Support

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

Country Code	Supported Code Pages
BE	437, 850
BI	437, 850
CF	863, 850
DK	865, 850
FR	437, 850
FB	437, 850
GR	437, 850
IT	437, 850
LA	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

Figure G-1. Supported Code Pages

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

Notes

Index

G

H

I

K

L

T

W

International Business
Machines Corporation
P.O. Box 1328-W
Boca Raton,
Florida 33429-1328

Printed in the
United States of America

74X9879