Personal Communications AS/400 and 3270
for Windows and DOS (Full-Function Mode)

IBM

Programmer's Guide

Version 4.1

SC31-8266-00

Personal Communications AS/400 and 3270
for Windows and DOS (Full-Function Mode)

IBM

Programmer's Guide

Version 4.1

© ODBC is copyrighted by Microsoft Corporation.

SC31-8266-00

Note

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

First Edition (March 1996)

This edition applies to Version 4.1 of Personal Communications for Windows and DOS (Full-Function Mode). It applies to all subsequent releases and modifications until otherwise indicated in new editions or technical newsletters. This edition is a major revision of SC18-2286, which still applies to Version 4.0 of this product. Make sure you are using the correct edition for the level of the product.

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

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

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

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

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

Contents

EHLLAPI Functions by Function Number xi

Notices xv

About This Book xvii
Who Should Read This Book xvii
How This Book Is Organized xvii
Highlighting Conventions xviii
Requirements xx
Hardware Requirements xx
Software Requirements xx
Host Requirements xxi
Compatibility with Other Emulation Products (For PC/3270) xxi
Related Publications xxi
Notation xxii

Chapter 1. Introduction 1
EHLLAPI/DDE Overview 1
Function Enhancement in PC/3270 Windows Mode 3
EHLLAPI Interface Description 3
Function Calls 4
Language Interface Modules and Dynamic Link Libraries 5
Getting Started 5

Chapter 2. Scenarios for Operating the Host System 7
Background 7
Scenario 1. Using the Search Function 8
Scenario 2. Sending Keystrokes 9
Scenario 3. Specifying Strings Using EHLLAPI 9
Indicating the Length of Strings 9
Steps 10
Scenario 4. Using Distributed Processing 10
Scenario 5. Transferring Files 11
Scenario 6. Automating Keystrokes 11
Scenario 7. Filtering Keystrokes 12
Scenario 8. Enhancing the Keyboard 12

Chapter 3. EHLLAPI Programming Overview 15
General 15
Using Language Interface Modules for PC/3270 DOS Mode 15
Customized Language Interface Modules 15
Provided Language Interface Modules 15
Source Code 16
Initialization 17
Memory Preallocation 17
Passing Call Parameter Values 18
Call Statement 19
Receiving Return Parameter Values 21
Exiting Applications 22
Compiler/Assembler/Interpreter 22

© Copyright IBM Corp. 1996

Linker . . . 22
Program Execution . . . 22
Debugging . . . 23

Chapter 4. EHLLAPI Programming Guidelines . . . 25
Summary of Presentation Space Specification . . . 25
Type of Presentation Space . . . 25
Size of Presentation Space . . . 25
Presentation Space Names . . . 25
Host-Connected Presentation Space . . . 26
PSID Handling . . . 26
Summary of Prerequisite Calls . . . 28
Summary of Return Codes . . . 30
Considerations for Using Send File (90) and Receive File (91) Functions (PC/3270 DOS Mode Only) . . . 31
Using the DOS SETBLOCK Command . . . 31
System File Preparation . . . 31
SEND.EXE and RECEIVE.EXE Files . . . 32
ASCII Mnemonics . . . 32
General . . . 33
Get Key (51) Function . . . 34
Send Key (3) Function . . . 35

Chapter 5. EHLLAPI Language Syntax and Considerations . . . 37
IBM BASIC (Compiler) Guidelines . . . 38
Source Code Preparation . . . 38
Compiling . . . 38
Linking . . . 38
Considerations for using IBM BASIC (Compiler) . . . 38
IBM BASIC (Interpretive) Guidelines . . . 39
Source Code Preparation . . . 39
Compiling . . . 39
Linking . . . 39
Considerations for Using IBM BASIC (Interpretive) . . . 39
IBM C Guidelines . . . 41
Source Code Preparation . . . 41
Compiling . . . 41
Linking . . . 41
IBM COBOL Guidelines . . . 42
Source Code Preparation . . . 42
Compiling . . . 42
Linking . . . 42
IBM Macro Assembler Guidelines . . . 43
Source Code Preparation . . . 43
EHLLAPI Function Call Syntax . . . 45
Assembling . . . 45
Linking . . . 45
IBM Pascal Guidelines . . . 45
Source Code Preparation . . . 45
Compiling . . . 46
Linking . . . 46
Considerations for Using Pascal . . . 46
C for Windows Guidelines . . . 48
Source Code Preparation . . . 48

Compiling . . . 49
Linking . . . 49
Macro Assembler for Windows Guidelines . . . 49
Source Code Preparation . . . 49
EHLLAPI Function Call Syntax . . . 50
Assembling . . . 50
Linking . . . 50

Chapter 6. EHLLAPI Functions . . . 51
Page Layout Conventions . . . 51
Prerequisite Calls . . . 51
Call Parameters . . . 51
Return Parameters . . . 51
Notes on Using This Function . . . 51
Running the EHLLAPI Sampler (PC/3270 DOS Mode Only) . . . 52
Loading and Running the Program Sampler . . . 52
Summary of EHLLAPI Functions . . . 53
Allocate Communications Buffer (123) . . . 55
Change PS Window Name (106) . . . 57
Change Switch List LT Name (105) . . . 59
Connect for Structured Fields (120) . . . 61
Connect Presentation Space (1) . . . 63
Connect Window Services (101) . . . 65
Convert Position or Convert RowCol (99) . . . 66
Copy Field to String (34) . . . 69
Copy OIA (13) . . . 73
Copy Presentation Space (5) . . . 81
Copy Presentation Space to String (8) . . . 85
Copy String to Field (33) . . . 89
Copy String to Presentation Space (15) . . . 91
Disconnect from Structured Fields (121) . . . 93
Disconnect Presentation Space (2) . . . 94
Disconnect Window Service (102) . . . 95
Find Field Length (32) . . . 96
Find Field Position (31) . . . 98
Free Communications Buffer (124) . . . 100
Get Key (51) . . . 101
Get Request Completion (125) . . . 105
Pause (18) . . . 108
Post Intercept Status (52) . . . 110
Query Close Intercept (42) . . . 111
Query Communications Buffer Size (122) . . . 112
Query Cursor Location (7) . . . 114
Query Field Attribute (14) . . . 115
Query Host Update (24) . . . 117
Query Session Status (22) . . . 118
Query Sessions (10) . . . 120
Query System (20) . . . 122
Query Window Coordinates (103) . . . 124
Read Structured Fields (126) . . . 126
Receive File (91) . . . 130
Release (12) . . . 133
Reserve (11) . . . 134
Reset System (21) . . . 135

Search Field (30) . . . 136
Search Presentation Space (6) . . . 138
Send File (90) . . . 140
Send Key (3) . . . 143
Keyboard Mnemonics . . . 145
Set Cursor (40) . . . 150
Set Session Parameters (9) . . . 151
Start Close Intercept (41) . . . 157
Start Host Notification (23) . . . 159
Start Keystroke Intercept (50) . . . 161
Start Playing Macro (110) . . . 164
Stop Close Intercept (43) . . . 165
Stop Host Notification (25) . . . 166
Stop Keystroke Intercept (53) . . . 167
Storage Manager (17) . . . 168
Get Storage . . . 169
Free Storage . . . 170
Query Free Storage . . . 171
Free All Storage . . . 172
Wait (4) . . . 173
Window Status (104) . . . 174
Write Structured Fields (127) . . . 177

Chapter 7. PCSAPI Functions . . . 181
How to Use PCSAPI . . . 181
Page Layout Conventions . . . 181
Function Type . . . 181
Parameter Type and Description . . . 181
Return Code . . . 181
pcsQueryEmulatorStatus . . . 182
pcsQueryWorkstationProfile . . . 183
pcsStartSession . . . 184
pcsStopSession . . . 185

Chapter 8. DDE Functions . . . 187
PC/3270 and PC400 DDE Data Items . . . 187
Using System Topic Data Items . . . 188
Using Session Topic Data Items . . . 189
Using LU Topic Data Items (PC/3270 Only) . . . 189
DDE Functions . . . 190
Naming Conventions for Parameters . . . 191
Code Conversion . . . 192
Find Field . . . 193
Get Keystrokes . . . 195
Get Mouse Input . . . 197
Get Number of Close Requests . . . 200
Get Operator Information Area . . . 201
Get Partial Presentation Space . . . 202
Get Presentation Space . . . 205
Get Session Status . . . 208
Get System Configuration . . . 210
Get System Formats . . . 212
Get System Status . . . 213
Get System SysItems . . . 214

Get System Topics . . . 215
Get Trim Rectangle . . . 216
Initiate Session Conversation . . . 217
Initiate Structured Field Conversation . . . 218
Initiate System Conversation . . . 219
Put Data to Presentation Space . . . 220
Search for String . . . 221
Send Keystrokes . . . 222
Session Execute Macro . . . 223
Set Cursor Position . . . 230
Set Mouse Intercept Condition . . . 231
Set Presentation Space Service Condition . . . 234
Set Session Advise Condition . . . 236
Set Structured Field Service Condition . . . 238
Start Close Intercept . . . 239
Start Keystroke Intercept . . . 241
Start Mouse Input Intercept . . . 242
Start Read SF . . . 246
Start Session Advise . . . 249
Stop Close Intercept . . . 251
Stop Keystroke Intercept . . . 252
Stop Mouse Input Intercept . . . 253
Stop Read SF . . . 254
Stop Session Advise . . . 255
Terminate Session Conversation . . . 256
Terminate Structured Field Conversation . . . 257
Terminate System Conversation . . . 258
Write SF . . . 259
Summary of DDE Functions . . . 260
DDE Menu Item API . . . 264
DDE Menu Client . . . 264
DDE Menu Server . . . 265
DDE Menu Functions . . . 266
Change Menu Item . . . 267
Create Menu Item . . . 273
Initiate Menu Conversation . . . 275
Start Menu Advise . . . 276
Stop Menu Advise . . . 278
Terminate Menu Conversation . . . 279
Summary of DDE Menu Item API Functions . . . 280

Chapter 9. Using DDE Functions with DDE Client Application (Visual Basic) . . . 281
Using PC/3270 and PC400 DDE Interface . . . 281
System Conversation . . . 282
Session Conversation . . . 283
Session Conversation (Hot Link) . . . 284
PC/3270 and PC400 DDE Interface . . . 286
DDE Functions for System Conversation . . . 287
Initiate System Conversation . . . 287
Get System Configuration . . . 287
Get System Formats . . . 288
Get System Status . . . 288
Get System SysItems . . . 288

Get System Topics . . . 289
Terminate System Conversation . . . 289
DDE Functions for Session Conversation . . . 290
Initiate Session Conversation . . . 290
Find Field . . . 290
Get Operator Information Area . . . 292
Get Partial Presentation Space . . . 293
Get Presentation Space . . . 293
Get Session Status . . . 294
Get Trim Rectangle . . . 294
Put Data to Presentation Space . . . 295
Search for String . . . 296
Session Execute Macro . . . 296
Set Cursor Position . . . 297
Terminate Session Conversation . . . 297
DDE Function for Session Conversation (Hot Link) . . . 298
Initiate Session Conversation . . . 298
Start Close Intercept . . . 298
Start Keystroke Intercept . . . 299
Start Session Advise . . . 299
Stop Close Intercept . . . 301
Stop Keystroke Intercept . . . 301
Stop Session Advise . . . 301
Terminate Session Conversation . . . 302
Visual Basic Sample Program . . . 303

Chapter 10. DOS Mode EHLLAPI Support in a Windows DOS Prompt for PC/3270 Version 1.01/2.0 for DOS . . . 315

Chapter 11. Server Requester Programming Interface Support . . . 319
SRPI API Interface . . . 319

Appendix A. EHLLAPI Messages . . . 321
COMHLL01 3270PCS is not installed . . . 321
COMHLL02 PCSHLL is already installed . . . 321
COMHLL03 Illegal parameter, PCSHLL is not installed . . . 321
COMHLL04 PCSHLL cannot open link with 3270PCS . . . 321

Appendix B. EHLLAPI Customized Language Interface Modules . . . 323
Functions a LIM Can Perform . . . 323
Revised Call Support for the Personal Computer . . . 325
Writing A Language Interface Module . . . 325

Appendix C. EHLLAPI Sample Programs . . . 327
IBM BASIC Sample Program . . . 327
IBM C Sample Program . . . 335
IBM COBOL Sample Program . . . 341
IBM Pascal Compiler Sample Program . . . 349

Appendix D. DDE Sample Program . . . 357
Creating a DDE Program to Support the Receive Load Button . . . 357
LISTFILE.C File . . . 359
LISTDDE.C File . . . 368
LISTFILE.H File . . . 381

LISTDATA.H File 384
LISTFILE.DEF File 385
LISTFILE.RC File 386

Appendix E. EHLLAPI Migration 387
Migrating HLLAPI Applications from DOS to Windows 387
DOS-Mode EHLLAPI Compatibility with Other Emulation Products 387
IBM Workstation Program HLLAPI 387
Additional Functions 387
Functions not Supported 388
Other Considerations 388
IBM Entry Emulator HLLAPI 388
Additional Functions 389
Other Considerations 389
IBM OS/2 Extended Edition HLLAPI 389
Considerations 389

Appendix F. Query Reply Data Structures Supported by EHLLAPI . . . 391
The DDM Query Reply 391
DDM Application Name Self-Defining Parameter 392
PCLK Protocol Controls Self-Defining Parameter 392
Base DDM Query Reply Formats 392
The IBM Auxiliary Device Query Reply 394
Direct Access Self-Defining Parameter 396
PCLK Protocol Controls Self-Defining Parameter 396
The OEM Auxiliary Device Query Reply 396
Direct Access Self-Defining Parameter 397
PCLK Protocol Controls Self-Defining Parameter 397
The Cooperative Processing Requester Query Reply 398
The Product-Defined Query Reply 398
Optional Parameters 398
Direct Access Self-Defining Parameter 399
The Document Interchange Architecture Query Reply 400
Direct Access Self-Defining Parameter 400

Index 403

EHLLAPI Functions by Function Number

The following list shows the page numbers where EHLLAPI functions are explained in this book.

1 "Connect Presentation Space (1)" on page 63
2 "Disconnect Presentation Space (2)" on page 94
3 "Send Key (3)" on page 143
4 "Wait (4)" on page 173
5 "Copy Presentation Space (5)" on page 81
6 "Search Presentation Space (6)" on page 138
7 "Query Cursor Location (7)" on page 114
8 "Copy Presentation Space to String (8)" on page 85
9 "Set Session Parameters (9)" on page 151
10 "Query Sessions (10)" on page 120
11 "Reserve (11)" on page 134
12 "Release (12)" on page 133
13 "Copy OIA (13)" on page 73
14 "Query Field Attribute (14)" on page 115
15 "Copy String to Presentation Space (15)" on page 91
17 "Storage Manager (17)" on page 168
18 "Pause (18)" on page 108
20 "Query System (20)" on page 122
21 "Reset System (21)" on page 135
22 "Query Session Status (22)" on page 118
23 "Start Host Notification (23)" on page 159
24 "Query Host Update (24)" on page 117
25 "Stop Host Notification (25)" on page 166
30 "Search Field (30)" on page 136
31 "Find Field Position (31)" on page 98
32 "Find Field Length (32)" on page 96
33 "Copy String to Field (33)" on page 89
34 "Copy Field to String (34)" on page 69
40 "Set Cursor (40)" on page 150
41 "Start Close Intercept (41)" on page 157
42 "Query Close Intercept (42)" on page 111
43 "Stop Close Intercept (43)" on page 165
50 "Start Keystroke Intercept (50)" on page 161
51 "Get Key (51)" on page 101
52 "Post Intercept Status (52)" on page 110
53 "Stop Keystroke Intercept (53)" on page 167
90 "Send File (90)" on page 140
91 "Receive File (91)" on page 130
99 "Convert Position or Convert RowCol (99)" on page 66
101 "Connect Window Services (101)" on page 65
102 "Disconnect Window Service (102)" on page 95
103 "Query Window Coordinates (103)" on page 124
104 "Window Status (104)" on page 174
105 "Change Switch List LT Name (105)" on page 59
106 "Change PS Window Name (106)" on page 57
110 "Start Playing Macro (110)" on page 164
120 "Connect for Structured Fields (120)" on page 61
121 "Disconnect from Structured Fields (121)" on page 93
122 "Query Communications Buffer Size (122)" on page 112
123 "Allocate Communications Buffer (123)" on page 55
124 "Free Communications Buffer (124)" on page 100
125 "Get Request Completion (125)" on page 105
126 "Read Structured Fields (126)" on page 126

© Copyright IBM Corp. 1996

127 “Write Structured Fields (127)” on page 177

Figures

1. Keystroke Flow 12
2. Host Presentation Space Characters 74
3. The Parts of PC/3270 EHLLAPI 324

Tables

1. Provided Language Interface Modules 16
2. Summary of Prerequisite Calls for the EHLLAPI Functions 28
3. Explanations of Return Codes 30
4. EHLLAPI Functions 53
5. Mnemonics Using Alphabetic Uppercase 145
6. Mnemonics Using Number or Alphabetic Lowercase 145
7. Mnemonics Using @A and Alphabetic Uppercase 146
8. Mnemonics Using @A and @Number or @A and @Alphabetic Lowercase 147
9. Mnemonics Using @A and @Special Character 147
10. Mnemonics Using @S and @Alphabetic Lowercase 147
11. Mnemonics Using @S and @Alphabetic Lowercase 148
12. Character Keys 148
13. Mnemonics Using Special Character Keys 149
14. Naming Scheme for Data Items 187
15. Applications for PC/3270 and PC400 187
16. Topics for PC/3270 and PC400 188
17. SENDKEY Command List 226
18. DDE Function Summary 260
19. DDE Menu Item API Function Summary 280
20. Naming Scheme for Data Items 281
21. Topics for PC/3270 or PC400 282
22. DDM Query Reply Base Format 391
23. DDM Application Name SDP 392
24. DDM PCLK Auxiliary Device SDP 392
25. Base DDM Query Reply Format with Name and Direct Access SDPs . 393
26. Base DDM Query Reply Format with Direct Access and Name SDPs . 394
27. IBM Auxiliary Device Base Format with Direct Access SDP 395
28. IBM Auxiliary Device Direct Access SDP 396
29. IBM Auxiliary Device PCLK SDP 396
30. OEM Auxiliary Device Base Format with Direct Access SDP 397
31. OEM Auxiliary Device Direct Access SDP 397
32. IBM Auxiliary Device PCLK SDP 397
33. CPR Query Reply Buffer Format 398
34. IBM Product Defined Query Reply Base Format 399
35. Valid REFID and SSID Values for the IBM Product Defined Query Reply 399
36. IBM Product-Defined Direct Access SDP 400
37. IBM DIA Base Format 400
38. IBM Product Defined Direct Access SDP 401

© Copyright IBM Corp. 1996

Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program or service is not intended to state or imply that only IBM's product, program, or service may be used. Subject to IBM's valid intellectual property or other legally protectable rights, any functionally equivalent product, program, or service may be used instead of the IBM product, program, or service.

Windows is a trademark of Microsoft Corporation.

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

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

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

© Copyright IBM Corp. 1996

About This Book

This book provides programming information for the IBM Personal Communications and the IBM Personal Communications AS/400 for Windows (*PC/3270* and *PC400* respectively) High-Level Language Application Program Interface / Dynamic Data Exchange (EHLLAPI/DDE).

EHLLAPI/DDE is used with PC/3270 or PC400 to provide a way for users and programmers to access the host presentation space with a set of functions that can be called from an application program running in a personal computer session.

If you want only to get started using PC/3270 or PC400 or have no interest in using the programming interface, see the *Personal Communications AS/400 and 3270 Version 4.1 Up and Running for Windows*.

Who Should Read This Book

This book is intended for personal computer users and programmers who write application programs that use EHLLAPI/DDE functions.

A working knowledge of the Disk Operating System (DOS) and the Microsoft** Windows** is assumed. For information about DOS and Windows, refer to the list of publications under “Related Publications” on page xxi.

You also need to be familiar with using a host system from a terminal or from a personal computer with terminal emulation software.

This book assumes you are familiar with the language and the compiler that you are using. For information on how to write, compile, or link-edit programs, refer to “Related Publications” on page xxi for the appropriate references for the specific language you are using.

How This Book Is Organized

Chapter 1, “Introduction” provides an overview of PC/3270 and PC400 EHLLAPI/DDE and discusses parts of the interface.

Chapter 2, “Scenarios for Operating the Host System” contains information about automated operation of the host system using EHLLAPI/DDE functions in applications.

Chapter 3, “EHLLAPI Programming Overview” provides general information concerning development of applications that use EHLLAPI functions.

Chapter 4, “EHLLAPI Programming Guidelines” provides information about combining multiple EHLLAPI functions into tasks.

Chapter 5, “EHLLAPI Language Syntax and Considerations” provides information concerning syntax and general considerations for each supported high-level language.

Chapter 6, “EHLLAPI Functions” describes each PC/3270 and PC400 EHLLAPI function in detail and explains how to run the PC/3270 and PC400 EHLLAPI Program Sampler. The functions are arranged alphabetically by name.

© Copyright IBM Corp. 1996

Chapter 7, “PCSAPI Functions” describes the PCSAPI function supported in the PC/3270 Windows mode and PC400.

Chapter 8, “DDE Functions” provides information concerning Dynamic Data Exchange functions of PC/3270 and PC400.

Chapter 9, “Using DDE Functions with DDE Client Application (Visual Basic)” describes how to use DDE functions with other Windows application programs (Microsoft Visual Basic**, Microsoft Excel**, Microsoft Word**) with DDE client functions.

Chapter 10, “DOS Mode EHLLAPI Support in a Windows DOS Prompt for PC/3270 Version 1.01/2.0 for DOS” provides information about the support for DOS mode EHLLAPI for PC/3270 Version 1.01/Version 2.0 for DOS.

Chapter 11, “Server Requester Programming Interface Support” includes information about PC/3270 for Windows Version 3.1 support for SRPI functions.

Appendix A, “EHLLAPI Messages” explains the messages that might be generated while EHLLAPI/DDE is executing.

Appendix B, “EHLLAPI Customized Language Interface Modules” describes how to write a Language Interface Module (LIM) for any computer language beyond those languages supported by PC/3270 EHLLAPI.

Appendix C, “EHLLAPI Sample Programs” provides sample programs that call PC/3270 EHLLAPI functions.

Appendix D, “DDE Sample Program” provides sample programs that call PC/3270 DDE functions.

Appendix E, “EHLLAPI Migration” lists considerations for adapting HLLAPI programs written for other IBMHLLAPI products for use with PC/3270 EHLLAPI.

Appendix F, “Query Reply Data Structures Supported by EHLLAPI” includes reference information about the query reply data structures that are supported by EHLLAPI.

Highlighting Conventions

Highlighting refers to different type styles used within text to emphasize certain kinds of information. Type styles other than the standard body text font that are used in this manual are shown along with a description of the kind of information that each different type style is used to emphasize.

Type style	**Emphasized Information**
CAPITAL LETTERS	CAPITAL LETTERS are used to emphasize commands, keywords, or other information that the programmer is to type in literally. This type style is not used in lengthy sample program listings.
Initial Caps	Initial uppercase letters emphasize EHLLAPI function names. These names are labels used to refer to the function. They are not used in an application to call an EHLLAPI function. The *function code* parameter is used to specify which EHLLAPI function is to be performed. See Chapter 6, “EHLLAPI Functions.”

Italics — When *italics* or ***bold italics*** appear in text, either a term is being used for the first time or the meaning implied by the word is being stressed.

Monospace — Monospace type is used in the following situations:

- In sample code listings of more than a few lines.
- As options for link or compile commands.
- For names of session options set by the Set Session Parameters (9) function. See "Set Session Parameters (9)" on page 151.
- As variable names (parameters) for a call statement to EHLLAPI functions. Note that you can use variable names other than those used in examples; however, you cannot substitute literal values for the variable names in a call to an EHLLAPI function.
- For ASCII mnemonics in text.
- For displayed information.
- For input from either a programmer or an operator.

Requirements

PC/3270 and PC400 EHLLAPI have the following hardware, software, and host computer requirements.

Hardware Requirements

In PC/3270 DOS mode, PC/3270 EHLLAPI/DDE requires approximately 31K bytes of storage in addition to the storage requirements for DOS and the PC/3270 Program. The product is designed to operate on the following hardware:

IBM Personal Computer
IBM Personal Computer AT
IBM Personal Computer XT
IBM Personal System/2
IBM Personal System/ValuePoint
IBM ThinkPad
Non-IBM PCs that are compatible with the IBM ones listed above

In PC/3270 Windows mode, PC/3270 EHLLAPI/DDE requires at least 1 MB of storage and is designed to operate on IBM Personal Computer with 80286 or a higher processor.

In PC400, PC400 EHLLAPI/DDE requires at least 1 MB of storage, and is designed to operate on IBM personal computers with a 80386** or higher processor.

Software Requirements

EHLLAPI/DDE has these software requirements:

DOS Versions 5.0 and later are supported.

Microsoft Windows** Version 3.1 (PC/3270 Windows mode or PC400).

PC/3270 or PC400 Version 4.1 Program is supported.

In PC/3270 DOS mode, write your PC/3270 EHLLAPI program in one of the following high-level languages (or equivalent):

- IBM Compiled BASIC 1.0, 2.0
- IBM BASIC Compiler/2
- IBM Interpretive BASIC 3.3
- IBM C 1.0
- IBM C/2 1.0
- IBM COBOL 2.0
- IBM COBOL/2
- IBM Pascal 2.02
- IBM Pascal Compiler/2
- IBM Macro Assembler
- IBM Macro Assembler/2

In PC/3270 Windows mode or PC400, you must write your PC/3270 or PC400 EHLLAPI/DDE program in a language supported by Microsoft Windows Software Development Kit.

Host graphics are not supported by PC/3270 EHLLAPI/DDE.

For additional information on available hardware and software that can be used with this product, refer to:

Personal Communications Version 4.1 Up and Running for DOS (Full-Function Mode)

Personal Communications Version 4.1 Reference for DOS (Full-Function Mode)

Personal Communications Version 4.1 Reference for Windows

Personal Communications AS/400 and 3270 Version 4.1 Up and Running for Windows

Host Requirements

EHLLAPI/DDE has the same host requirements as PC/3270 or PC400. Refer to:

Personal Communications Version 4.1 Up and Running for DOS (Full-Function Mode)

Personal Communications Version 4.1 Reference for DOS (Full-Function Mode)

Personal Communications Version 4.1 Reference for Windows

Personal Communications AS/400 and 3270 Version 4.1 Up and Running for Windows

Compatibility with Other Emulation Products (For PC/3270)

Most EHLLAPI applications that are written for the following IBM HLLAPI products can be used with PC/3270 EHLLAPI. Specific information needed to adapt applications written for these IBM HLLAPI products for use with PC/3270 EHLLAPI is in Appendix E, “EHLLAPI Migration” on page 387.

IBM Workstation Program HLLAPI
IBM Entry Emulator HLLAPI
IBM OS/2 Extended Edition (OS/2 EE) Communications Manager HLLAPI

Related Publications

Personal Communications Version 4.1 Up and Running for DOS (Full-Function Mode), SC31-8262

Personal Communications Version 4.1 Reference for DOS (Full-Function Mode), SC31-8265

Personal Communications AS/400 and 3270 Version 4.1 Up and Running for Windows, SC31-8261

Personal Communications AS/400 and 3270 Version 4.1 Reference for Windows, SC31-8264

IBM Disk Operating System User's Guide, SA40-0634

IBM Disk Operating System Technical Reference, S52G-9932

Microsoft Windows Software Development Kit

Microsoft Windows User's Guide

IBM 3270 Information Display System Data Stream Programmer's Reference, GD21-0028

IBM 5250 Information Display System Functions Reference Manual, SA21-9247

Notation

A table at the beginning of each section explains API or DDE functions in Chapter 6, “EHLLAPI Functions,” Chapter 7, “PCSAPI Functions,” and Chapter 8, “DDE Functions.” It shows the products that provide the function described in the section. ‘X’ means it is supported, and ‘–’ means not supported. For example, the following table indicates that a function is available in the PC/3270 Windows mode and PC400; it is not supported in the PC/3270 DOS mode.

PC/3270		*PC400*
DOS	Windows	
–	X	X

Chapter 1. Introduction

This chapter provides an overview of PC/3270 and PC400 EHLLAPI/DDE and discusses parts of the interface.

EHLLAPI/DDE Overview

The PC/3270 or PC400 EHLLAPI/DDE lets you use personal computer programs to interact with your host session.

In PC/3270 DOS mode, the following languages are supported for the personal computer programs.

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

In PC/3270 Windows mode or PC400, you can write personal computer programs using PC/3270 or PC400 EHLLAPI/DDE in the languages supported by Microsoft Windows Software Development Kit.

EHLLAPI/DDE works in a variety of environments. The interaction between host computers and personal computers made possible by EHLLAPI/DDE can increase productivity for experienced users and provide a shorter learning curve for inexperienced users. These results can be achieved by using EHLLAPI/DDE functions in personal computer application programs to:

Automate repetitive tasks.

Mask complex applications from the user.

Consolidate several complicated tasks into one simple task.

Simplify existing host applications.

© Copyright IBM Corp. 1996

Provide for unattended operation through a ***pogram operator*** monitoring tasks without human intervention. This pogram operator can:

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

Create ***composite screen*** applications using input from the host. The input for a composition screen is formatted and presented within a personal computer presentation space.

Write programs that divide the work between host and personal computer sessions. EHLLAPI/DDE lets you write and use an application to access files and programs in either the host or the personal computer session. Some tasks such as file editing can be done in the personal computer session; more demanding tasks (database searches, updates, and retrievals) can be done in the host session.

In other words, you can use EHLLAPI/DDE with PC/3270 or PC400 functions to:

Enhance interaction between a personal computer application program and a host session

Simplify your programming tasks.

In PC/3270 Windows mode and PC400, you can use both EHLLAPI and DDE interfaces. In PC/3270 DOS mode, you can use EHLLAPI interface. Note that personal computer programs written as DOS application programs run under the DOS mode EHLLAPI and do not work under the Windows mode EHLLAPI/DDE. Personal computer programs written as Windows application programs run under the Windows mode EHLLAPI/DDE and do not work under DOS mode EHLLAPI.

Function Enhancement in PC/3270 Windows Mode

The following APIs are enhanced in PC/3270 Windows mode Version 4.1. PCSAPI is a new set of programming interfaces for controlling PC/3270 sessions.

EHLLAPI

- Start Playing Macro (110)
- Asynchronous mode support
  - Start Host Notification (23)
  - Start Close Intercept (41)
  - Start Keystroke Intercept (50)
  - Read Structured Fields (126)
  - Write Structured Fields (127)

PCSAPI

- Start Session
- Stop Session
- Query Workstation
- Query Emulator Status

DDE

- Get Session Status. Returns the window handle of the session window.
- Find Fields. Searches a field by specifying its field type.
- Mouse input intercept functions
  - Start Mouse Input Intercept
  - Stop Mouse Input Intercept
  - Get Mouse Input
  - Set Mouse Intercept Condition
- New user interface formats such as Visual Basic** for the following functions:
  - Find field
  - Get partial presentation space
  - Get trim rectangle
  - Put data to presentation space
  - Search for string
  - Start for string
  - Start session advise

EHLLAPI Interface Description

The interface between an EHLLAPI application and the host has these features:

PC/3270 or PC400 software allows your personal computer to interface with a host by emulating a terminal.

PC/3270 software includes language interface modules (LIM) for DOS mode application programs. PC/3270 and PC400 software include dynamic link libraries (DLL) for Windows mode application programs to provide a way to interface between an application program and the terminal emulation software.

Function calls pass four or five parameters to a LIM or DLL which in turn, passes them to the terminal emulation software.

The terminal emulation software passes the data on to the host. To the host, this data looks just like it came from a terminal.

The terminal emulation software receives the host response and passes the result back to the LIM or DLL.

The LIM or DLL then converts the data into the required parameters and passes them back to the application.

To the user, this entire process is transparent. For this work to happen, you need to write, compile, and link an application with valid function calls. A valid function call in the correct syntax provides valid data for the required parameters and uses the appropriate entry point name for the LIM with which the application is linked. For a DLL, the entry point name for function calls is **HLLAPI** or **HLLAPIW**.

Function Calls

Depending on the language in which an application is written, an EHLLAPI function call has three elements:

A CALL verb
A LIM/DLL entry-point name
Four or five parameters

The CALL verb and LIM entry point name are not required in some languages. All four or five parameters must be passed by reference even though some functions do not use all four or five parameters. If a parameter is not used by a function, the variable name value is ignored.

The four or five parameters that are passed are the:

window handle
function code
data string
length
return code parameters.

On call and return, the function code, length, and return code parameters are 2-byte integers in all supported languages except IBM Macro Assembler. (In IBM Macro Assembler language applications, these values are 1-byte binary integers.) On call and return, the data string parameter contains string (nonnumeric) data. Numbers in the data string are handled as string data, not numeric data.

EHLLAPI is driven by function codes. This means that each EHLLAPI function has a number (function code) associated with it. This number (function code parameter) is used when calling the PC/3270 or PC400 EHLLAPI function from within an application program.

As a minimum, all functions require a function code on call, and all functions return a function code and a return code parameter. The function code is always returned unchanged. For all the functions except Convert Position or RowCol (99) function), the return code value on return indicates the status of the function or of the host. Information concerning return parameters and required call parameters for each function is contained in Chapter 6, “EHLLAPI Functions.”

The return code parameter indicates whether the function executed successfully and, in some cases, explains the failure. For example, if an incorrect function code

is specified, a return code of 10 is returned. Refer to “Summary of Return Codes” on page 30 for more information.

The data string and length parameters are discussed for each function in Chapter 6, “EHLLAPI Functions.”

More specific information concerning syntax and content of function calls is provided in “Source Code” on page 16. Sample source code listings that include actual calls to PC/3270 EHLLAPI for DOS mode are in Appendix C, “EHLLAPI Sample Programs” on page 327.

Language Interface Modules and Dynamic Link Libraries

A language interface module (LIM) provides an interface between your application and the PC/3270 EHLLAPI software in the DOS mode. A dynamic link library (DLL) provides an interface between your application and the PC/3270 or PC400 EHLLAPI/DDE software in the Windows mode. LIM/DLL is necessary because each high-level language has a unique method of storing data and calling subroutines. LIMs allow PC/3270 EHLLAPI to support several high-level languages.

A LIM is not required for applications written in Macro Assembler language. LIMs for the other supported languages are provided as part of PC/3270 software. Refer to “Using Language Interface Modules for PC/3270 DOS Mode” on page 15 for more information on LIMs.

DLLs are part of PC/3270 or PC400 software. Applications in the Windows mode are required to call a DLL for using EHLLAPI functions.

If you write your application in an unsupported language or if you want to limit the number of PC/3270 EHLLAPI functions available to your programmers, a customized LIM might be required. Refer to Appendix B, “EHLLAPI Customized Language Interface Modules” on page 323 for more information on customized LIMs.

Note: For Windows applications, the customized LIM is not supported.

Getting Started

Before using PC/3270 or PC400 EHLLAPI, you need to customize your personal computer for use with EHLLAPI. Refer to:

Personal Communications Version 4.1 Up and Running for DOS (Full-Function Mode)

Personal Communications Version 4.1 Reference for DOS (Full-Function Mode)

Personal Communications Version 4.1 Up and Running for Windows

Personal Communications Version 4.1 Reference for Windows

Personal Communications AS/400 and 3270 Version 4.1 Up and Running for Windows

To use EHLLAPI, you need an application program that has calls to EHLLAPI functions (see “Function Calls” on page 4) and is written in a supported high-level programming language (see “Software Requirements” on page xx). If you do not have such an application, try the Program Sampler (PCSSAMP.EXE) on the

PC/3270 diskette. See “Running the EHLLAPI Sampler (PC/3270 DOS Mode Only)” on page 52 for more information.

For more information about running an application, see “Program Execution” on page 22.

Chapter 2. Scenarios for Operating the Host System

This chapter contains information about automated operation of the host system using EHLLAPI/DDE functions in applications. Each scenario describes tasks that involve multiple EHLLAPI functions or interaction between the host and the personal computer application. Using EHLLAPI to automate these tasks, you can reduce or eliminate the need for operator intervention. The general concepts described in this chapter also apply to DDE.

Background

To write an effective EHLLAPI program, think of it as a programmed terminal operator. Look at actions a terminal operator would perform and compare them to actions that you can emulate with your EHLLAPI program operator.

When terminal operators sit down at a workstation, they typically look at the screen, determine where they are in the application and take the necessary actions to get to the desired starting point. This can involve an action as simple as pressing the Clear key, or it might require routing through an APPN network to the proper application.

When you write an EHLLAPI application program, your “program operator” (the software user that performs and monitors activities in your IBM 3270 Workstation or 5250 Workstation without actual human intervention) performs these user functions.

The sample scenarios presented here are intended to provide conceptual information about activities that can be facilitated by using EHLLAPI. Scenarios are presented that deal with the duties that your EHLLAPI program operator can perform in these areas:

Host System Operation, including

- Searching
- Sending keystrokes
- Specifying string

Distributed Processing, including

- Data extraction
- File transfer

Integrating interfaces

© Copyright IBM Corp. 1996

Scenario 1. Using the Search Function

There are four phases in a typical host system transaction:

1. Starting the transaction
2. Waiting for the host system to respond
3. Analyzing the response to see if it is the expected response
4. Extracting and using the data from the response

Your program operator can use a series of EHLLAPI functions to mimic these actions. After determining the correct starting point for the host system transaction, the program operator can call the Search Presentation Space (6) function to determine which keyword messages or prompting messages are on the display screen.

Next, the program operator can use the Send Key (3) function to key data into a host system session and enter a host system transaction. Then the program operator can:

Use the Wait (4) function that waits for the X CLOCK, X [], or X SYSTEM condition to end (or returns a keyboard-locked condition if the terminal has locked up).

If the keyboard is inhibited, your EHLLAPI program can call the Copy OIA (13) function to get more information about the error condition.

Use the Search Presentation Space (6) function to look for an expected keyword to validate that the proper response had been received.

Use the Copy Presentation Space to String (8) function (or any of several data access functions) to extract the desired data.

The Search Presentation Space (6) function is critical to simulate another task of the terminal operator. Some host systems do not stay locked in X CLOCK, X [], or X SYSTEM mode until they respond. Instead, they quickly unlock the keyboard and allow the operator to stack other requests. In this environment, the terminal operator depends on another visual prompt to know that the data has returned (perhaps a screen title or label). The Search Presentation Space (6) function allows your EHLLAPI program to search the presentation space while waiting. While waiting for a response, calling the Pause (18) function allows other DOS sessions to share the processing unit resource. The Pause (18) function has an option that allows your EHLLAPI program to wait for a host system update event to occur.

If no host system event occurs after a reasonable timeout period, the EHLLAPI program could call a customized error message such as:

No Response From Host. Retry?

In this environment, program revisions are very important considerations since the program operator must be reprogrammed for even minor changes in the display messages. For example, if a terminal operator expects the message:

Enter Part Number:

as a prompt, he or she will probably be able to respond properly to an application change that produces the message:

Enter Component Number:

Since the program operator is looking for a literal keyword string, subtle changes in message syntax, even as trivial as uppercase versus lowercase, can make the program take a preprogrammed error action.

Scenario 2. Sending Keystrokes

There are several considerations requiring attention in designing programs that send keystrokes to the host system. In some application environments, issuing a command is as simple as keying in a string and pressing Enter. Other applications involve more complex IBM workstation formatted screens in which data can be entered into any one of several fields. In this environment, you need to understand the keystrokes required to fill in the display screen.

The Tab key mnemonic (@T, see page 34 for a full list of mnemonics) can be used to skip between fields. When sending keystrokes to a field using the Send Key (3) function, be aware of the field lengths and contents. If you fill the fields completely and the next attribute byte is “autoskip,” your cursor is then moved to the next field. If you then issue a tab, you skip to yet another field.

If your keystrokes do not completely fill the field, there might be data left from prior input. Use the Erase End of Field (EOF) command to clear this residual data.

Scenario 3. Specifying Strings Using EHLLAPI

String data is passed and returned by reference in the second parameter of the calling statement.

The calling data string can contain:

- A presentation space short name
- A command
- A search string
- Concatenated data items
- Data to be copied from one session to another

The return data string can contain:

- Concatenated data items
- Data to be copied from one session to another

Indicating the Length of Strings

EHLLAPI requires you to indicate the length of all strings.

The length of calling data strings can be specified by either of the two methods described in “Steps” on page 10, except for the Set Session Parameters (9) function.

For returned data strings, memory must be preallocated. The method used to do this depends on the high-level language in which your application is written. See “Memory Preallocation” on page 17 for more information.

Steps

To specify the length of calling data strings, do either of these steps:

Pass the length of each string in the length parameter of the calling statement

Include an end of text (EOT) character at the end of each string.

You select the method by entering either the STRLEN or STREOT option of the Set Session Parameters (9) function.

When the STRLEN option is in effect, the explicit number of bytes in the calling data string must be passed in the calling length parameter. This value is arrived at by manually counting the number of bytes in the calling data string.

Using the STREOT option avoids this manual calculation. When this option is in effect, you enter a character that you specify using the EOT=c option at the end of the calling data string. EHLLAPI uses this EOT character to delimit the string.

To send the following string to a session:

XREF 12345

followed by the Enter key, you can pass

XREF 12345

as the calling data string in the Copy String to Presentation Space (15) function and pass @E as the calling data string in the Send Key (3) function.

If the STRLEN option is in effect, you must pass 10 as the calling length parameter in your call to the Copy String to Presentation Space (15) function and 2 as the calling length parameter in your call to the Send Key (3) function.

If the STREOT option is in effect, you do not have to specify a value for the calling length parameter in either call. However, you must include the EOT character in the calling data string parameter for both calls. You could specify the exclamation mark as the EOT character by including EOT=! in a call to the Set Session Parameters (9) function. If so, the calling data strings required to send the preceding text become XREF 12345! and @E!.

Note: The preceding example could have been accomplished in a single call to the Send Key (3) function. The call to the Copy String to Presentation Space (15) function is used to improve performance. The Send Key (3) function must be used to send the Enter key mnemonic.

This EOT character can be used to delimit only calling data strings. However, the STREOT cannot be used to specify the length of the calling data string for the Set Session Parameters (9) function. For calls to EHLLAPI functions that return a data string, you must code for the exact length of the returned data string.

Scenario 4. Using Distributed Processing

There are a number of applications in the collaborative category. These applications provide a single end-user interface, but their processing is performed at two or more different physical locations.

An EHLLAPI application can interact with host system applications by intercepting the communications between the host system and the terminal user. The host

system presentation space is the vehicle used to intercept this data. The local application can request to be notified each time the presentation space is updated or whenever an AID key is pressed by the operator.

This personal computer application can then cooperate with a host system application in any of the following ways:

On a field or presentation space basis using either the copy functions that address fields (Copy String to Field (33) function or Copy Field to String (34) function) or the functions that let you copy from and into presentation spaces (for example, Copy String to Presentation Space (15) function or Copy Presentation Space to String (8) function).

On a keystroke basis, using the Send Key (3) function.

On a file basis for large blocks of data. You can have your application use the EHLLAPI file transfer capability (using Send File (90) function or Receive File (91) function) to transfer data or functions (such as load modules) and have it processed locally or remotely.

Scenario 5. Transferring Files

In this scenario, assume that you want to automate a file transfer:

You could begin by using the procedure discussed in the search scenario earlier to logon to a host system session.

Instead of using copy functions (which are inefficient for copying many screens of data), your EHLLAPI program could call file transfer functions Send File (90) and Receive File (91) to transfer data.

Upon successful completion:

- If the Send File (90) function finished executing, your EHLLAPI program could submit a batch job using either a copy function or the Send Key (3) function before logging off.
- If the Receive File (91) function finished executing, your EHLLAPI program could start up a local application.

Scenario 6. Automating Keystrokes

An application can provide all of the keystrokes for another application or can intersperse keystrokes to the target destination with those from the keyboard. Sometimes in order to do this, the application must &q.lock out= other sources of keystroke input that might be destined for a target application or presentation space (using the Reserve (11) function) and the later &q.unlock= it (using the Release (12) function).

The origin of keystrokes presented to any application is determined by the design of the application. Keystrokes can originate from:

The keyboard
Data integrated into the source application
Secondary storage retrieved through the DOS interface
The PC/3270 and PC400 interface

In all cases, the keystrokes provided to the target application are indistinguishable from the ordinary operator input.

Scenario 7. Filtering Keystrokes

An application acting as a filter can intercept a keystroke coming from EHLLAPI (either from the keyboard or a source application) that is targeted for another destination. The keystroke can be:

Ignored; that is, deleted
Redirected to another alternate application
Validated
Converted; for example, uppercase to lowercase
Enhanced through keyboard macros

Figure 1 represents the keystroke flow and the objects within a keyboard enhancement environment.

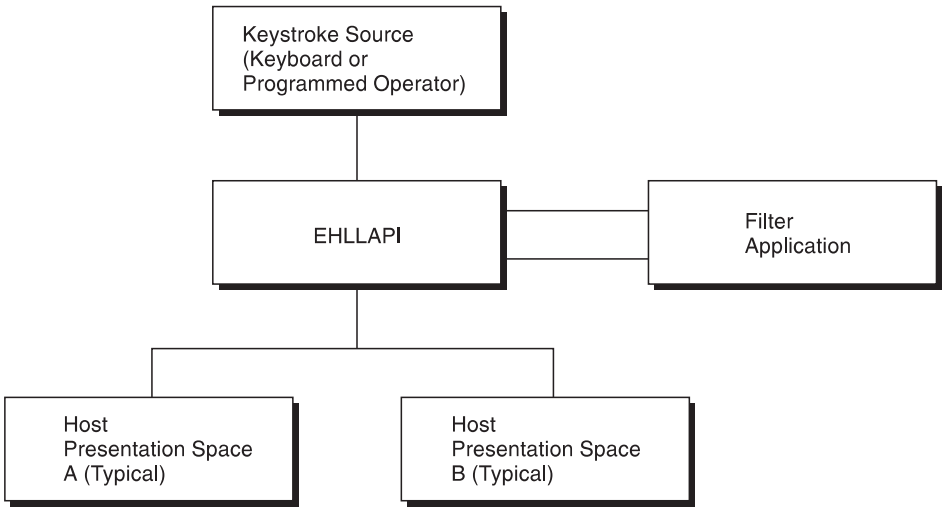


Figure 1. Keystroke Flow

Scenario 8. Enhancing the Keyboard

Scenario 8 uses filtering to create an *enhancer application program*. An enhancer application program monitors the data coming in from the keyboard and changes it in some specified way. Typically, these application programs use instructions called *keyboard macros* that tell them what keystrokes to look for and what changes to make. The change might involve suppressing a keystroke (so it appears to the target application as though it was never sent), replacing a keystroke with another, or replacing a single keystroke with a series of keystrokes.

To do this using EHLLAPI, you can construct this scenario:

1. Your EHLLAPI application program calls the Connect Presentation Space (1) function to connect to the presentation space whose keystrokes are to be filtered.

2. Your EHLLAPI program next calls the Start Keystroke Intercept (50) function specifying the L option. This causes all keystrokes to be routed to the filtering application program.
3. The filtering application program can now define a loop in which:
   a. The Get Key (51) function intercepts all keystrokes being sent to the target presentation space.
   b. The filtering application examines each keystroke and performs a keyboard macro task such as:

      Abbreviating program commands so that a three- or four-keystroke command can be condensed into one keystroke

      Customizing commands that are easier to remember or are consistent with other software packages

      Creating *boiler plates* for contracts or frequently used letters

      Rearranging the keyboard for concurrent applications that use the same keys for differing functions

      For example, the filtering application can convert a key combination, such as Alt + Y, into a command that positions the cursor at column 35 of the second line in presentation space and writes the string XYZ Tool Corporation, Dallas, Texas.

   c. If a keystroke is rejected, the EHLLAPI program can cause a warning to be sounded, using the Post Intercept Status (52) function.
4. After your EHLLAPI program exits the filtering loop, you can call the Stop Keystroke Intercept (53) function to end the filtering process.

Without EHLLAPI, you either need to purchase a keyboard enhancer product or use complicated assembler language and BIOS facilities to divert and inspect keystrokes. With EHLLAPI, you can perform these functions more easily and efficiently.

Chapter 3. EHLLAPI Programming Overview

Before writing a program that uses EHLLAPI functions, determine whether an application requires host interaction.

Once the functional requirements for a program have been defined and it is determined that host support is required or desirable, the programmer should identify those tasks that will be performed by the host. A set of calls to EHLLAPI functions necessary to perform the required tasks can then be incorporated in the application. Refer to Chapter 6, “EHLLAPI Functions” for information about each specific EHLLAPI function. Refer to Appendix C, “EHLLAPI Sample Programs” for examples of source code that contain calls to EHLLAPI functions in the PC/3270 DOS mode.

This chapter provides information needed to incorporate EHLLAPI functions into applications written in each supported high-level language. It also tells you how to run an application and how to use the trace facility for debugging purposes.

General

Programs using EHLLAPI functions are usually typed in as source code in a supported high-level language, compiled and linked. Then the executable file can be run with the host to test and debug the individual functions. The following paragraphs provide information about each of these steps.

Using Language Interface Modules for PC/3270 DOS Mode

The following paragraphs contain guidelines concerning the use of language interface modules (LIM).

Note: LIMs are prepared for the PC/3270 DOS mode EHLLAPI, and do not support the PC/3270 Windows mode EHLLAPI or the PC400 EHLLAPI.

Customized Language Interface Modules

A customized LIM is required only when your application program is written in an unsupported high-level language or when you want to limit the number of EHLLAPI functions available to the programmer. Though EHLLAPI is designed to allow the use of customized LIMs, IBM assumes no responsibility for the generation, support, or use of customized LIMs. If you use a customized LIM, use a file name different from those listed in Table 1 on page 16 for provided LIMs.

Refer to Appendix B, “EHLLAPI Customized Language Interface Modules” for more information concerning customized LIMs.

Provided Language Interface Modules

LIMs are provided as part of EHLLAPI software. Information about each provided LIM is in Table 1 on page 16.

Note: The two names listed in the third column of Table 1 on page 16 for each language are alternate names for the same LIM entry point. Calling either name produces the same results. Multiple names are supported to

© Copyright IBM Corp. 1996

eliminate the need to change entry point names in some migrating applications.

Table 1. Provided Language Interface Modules

Language	Filename	Called LIM Entry Point	Comments
IBM BASIC, Compiled	HLLCBAS.OBJ	HLLCBAS BLIM	
IBM BASIC, Interpretive	None. Linking is not required.	BLIM	This is a special LIM that is available anytime PC/3270 EHLLAPI is loaded.
IBM C	HLLC_C.OBJ	HLLC CLIM	This LIM is for compact model applications.
IBM C	HLLC_L.OBJ	HLLC CLIM	This LIM is for large model applications.
IBM C	HLLC_M.OBJ	HLLC CLIM	This LIM is for medium model applications.
IBM C	HLLC_S.OBJ	HLLC CLIM	This LIM is for small model applications.
IBM COBOL	HLLCOB.OBJ	HLLCOB COBLIM	
IBM Macro Assembler	None	None	The IBM Macro Assembler does not require a LIM.
IBM Pascal	HLLPAS.OBJ	HLLPAS PASLIM	

These LIMs are intended to reduce the amount of modification required to use an existing HLLAPI application written in a supported high-level language for Workstation Program HLLAPI or Entry Emulator HLLAPI. These LIMs cannot be used with applications written in an unsupported high-level language.

Source Code

The following paragraphs describe source code statements that are required to use EHLLAPI functions for applications.

Write PC/3270 DOS mode applications in:

IBM BASIC (Compiled)
IBM BASIC (Interpretive)
IBM C
IBM COBOL
IBM Macro Assembler
IBM Pascal

Note: *Some* of this information does not apply to IBM Macro Assembler. See "IBM Macro Assembler Guidelines" on page 43.

Windows mode applications should be written in a language supported by Microsoft Windows Software Development Kit.

Note: Follow also general rules described in *Microsoft Windows Software Development Kit* to write Windows applications.

Initialization

Generally, the variable names used in PC/3270 and PC400 EHLLAPI function call statements must be declared at some point before the call is executed. An exception to this rule is that for applications written in IBM BASIC, variable names that end in % or $ can be used instead of declaration statements.

IBM BASIC (Interpretive) requires that a set of statements shown under “Required Statements” on page 39 be executed before the first call to an EHLLAPI function.

For the PC/3270 DOS mode or the PC400 applications, it is a good idea to execute the Reset System (21) function at the beginning of your application to create a known set of conditions. Also, check the return code. A return code of 1 (PC/3270 DOS mode) or 9 (PC400) indicates that EHLLAPI is not loaded. You can include an operator prompt to indicate the remedy. The Query System (20) function also generates a return code of 1 (PC/3270 DOS mode) or 9 (PC400) if EHLLAPI is not loaded.

Specific initialization statements for each supported high-level language are described beginning on page 37.

Note: For the PC/3270 Windows mode applications, it might affect another EHLLAPI applications to execute the Reset System (21) function. Pay attention to the conflict among EHLLAPI applications, when you execute the Reset System (21) function in the Windows mode.

Memory Preallocation

PC/3270 and PC400 EHLLAPI functions that return a data string parameter require a statement that preallocates memory for the return data string value. The affected functions are:

Copy Field to String (34)
Copy OIA (13)
Copy Presentation Space (5)
Copy Presentation Space to String (8)
Get Key (51)
Query Session Status (22)
Query Sessions (10)
Query System (20)
Get Storage subfunction 01 of Storage Manager (17)

Note: Get Storage subfunction 01 is only for the PC/3270 DOS mode applications. Execute the Global Allocation function for PC/3270 Windows mode or PC400 instead.

IBM BASIC

To preallocate the required memory in IBM BASIC, execute a statement immediately prior to the call statement that sets the value of the variable used to pass the data string parameter to a value with the same length as the expected return data string. For example, the BASIC statement on page 334

```
dstring$ = SPACE$(35)
```

is used to preallocate the memory to receive the return data string for the Query System (20) function. The expected length of this data string is always 35 bytes. See “Query System (20)” on page 122.

IBM C

In IBM C, enter a statement like:

```
char q_buffer[35];              / for query system /
```

on page 336 to preallocate the required memory. This syntax is also applicable for the PC/3270 Windows mode EHLLAPI applications.

IBM COBOL

In IBM COBOL, enter a statement like:

```
1   4  -query-system-string.
```

and the 15 lines after it on page 342 to preallocate the required memory.

IBM Macro Assembler

In IBM Macro Assembler, include a statement like:

```
q_sys_str db 35 dup ( )
```

to preallocate the required memory. This syntax is also applicable for the PC/3270 Windows mode applications.

IBM Pascal

In IBM Pascal, include a statement like:

```
var
   q_sys_str : string(35);
```

on page 351 in your source code.

Passing Call Parameter Values

PC/3270 and PC400 EHLLAPI function calls pass parameter values by reference. Literal values for parameters cannot be entered in the calling statement. Prior to calling an EHLLAPI function, the value for each parameter required by the function must be stored in the corresponding variable name. Values need not be set for call parameters listed as NA in Chapter 6, “EHLLAPI Functions.”

EHLLAPI function calls must pass (by reference) the following four parameters in this fixed order:

1. Function number
2. Data string
3. Length or a buffer size
4. Host presentation space position

Function Number (first parameter): specifies the EHLLAPI function to be executed and is always required. For PC/3270 DOS mode applications, except for the applications written in IBM Macro Assembler language, this position must be filled by a 2-byte (fullword) integer. For PC/3270 Windows mode applications, this position must be filled by a 4-byte integer. For the PC400 applications, this position must be filled by a 4-byte long pointer that points to a 2-byte integer value.

Data String (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. When NA, this parameter can be any string.

Length (third parameter): is usually the length (in number of bytes) of the calling data string described in the preceding paragraph. In special cases, this parameter passes information such as buffer size. When used, this position must be filled by a 2-byte integer (PC/3270 DOS mode), a 4-byte integer (PC/3270 Windows mode), or a 4-byte long pointer that points to a 2-byte integer value (PC400). When the length is either NA or overridden by the current value of a session parameter defined by the Set Session Parameters (9) function, this integer can be any value.

Presentation Space (PS) Position (fourth parameter): is a value associated with the IBM 3278/79 or 5250 Display Station screen sizes emulated by the personal computer. When used, this position must be filled by a 2-byte integer (PC/3270 DOS mode), a 4-byte integer (PC/3270 Windows mode), or a 4-byte long pointer that points to a 2-byte integer value (PC400) between 1 and 3564. The valid upper limit for the value of this parameter can be less than 3564 depending on how you configure the host presentation space. See “Summary of Presentation Space Specification” on page 25.

When PS Position is either NA or overridden by the current value of a session parameter defined by the Set Session Parameters (9) function, this 2-byte integer can be any value.

Note: Window Handle of the application program must be passed followed by the four parameters when calling EHLLAPI functions of which entry point name is HLLAPIW (for PC/3270 Windows mode or PC400). See “C for Windows Guidelines” on page 48 and “Macro Assembler for Windows Guidelines” on page 49.

Call Statement

After required initialization statements are executed, calls to EHLLAPI functions can be located anywhere you need them within the application code. Except for applications written in IBM Macro Assembler, the pattern for an EHLLAPI function call is:

PC/3270 DOS Mode

```
CALL LIM_ENTRYPT(FUNC,DATA_STRING,PARM_3,PARM_4);
```

PC/3270 Windows Mode or PC400

```
CALL hllapi(FUNC,DATA_STRING,PARM_3,PARM_4)
CALL hllapiw(HWND,FUNC,DATA_STRING,PARM_3,PARM_4)
```

Term	Description
CALL	Reserved verb. Not required in some supported languages.
LIM_ENTRYPT	The LIM_ENTRYPT is different for each supported language. For provided LIMs, valid LIM entry points are listed under “Provided Language Interface Modules” on page 15. For customized LIMs, valid LIM entry points are specified by the user.
hllapi	hllapi is an entry point name of EHLLAPI's dynamic link libraries (HLLAPI.DLL). A user EHLLAPI application can connect only one session simultaneously.

hllapiw — hllapiw is an entry point name of EHLLAPI's dynamic link libraries (HLLAPI.DLL). By specifying a different window handle as the first parameter, user EHLLAPI applications can connect multiple sessions simultaneously.

HWND — Window handle of the calling EHLLAPI application, this field is a 2-byte integer.

FUNC — As the first parameter, is the EHLLAPI function number, always on call and on return. For PC/3270 DOS mode applications, the function number is expressed as a 2-byte integer. For PC/3270 Windows mode or PC400 applications, this field is a 4-byte long pointer that points to a 2-byte integer function number.

DATA_STRING — As the second parameter, contains presentation space identification, function options, or data on call or return. For the PC/3270 DOS mode applications, the data string is expressed as a data string. For PC/3270 Windows mode or PC400 applications, this field is a 4-byte long pointer that points to a string.

PARM_3 — As the third parameter, when used on call or return contains a data string length, the length of a field in the host presentation space or a host presentation space position. In Chapter 6, “EHLLAPI Functions,” the heading *Length* is used for information concerning use of this parameter during call and return. For PC/3270 DOS mode applications, the PARM-3 is expressed as a 2-byte integer. For PC/3270 Windows mode or PC400 applications, this field is a 4-byte long pointer that points to a 2-byte integer value.

PARM_4 — As the fourth parameter, is a 2-byte integer (PC/3270 DOS mode) or 4-byte long pointer (PC/3270 Windows mode or PC400) that points to a 2-byte integer value on call or return. When used on call, is usually the host presentation space position. Always used on return, and usually contains the return code. In Chapter 6, “EHLLAPI Functions,” the heading *PS Position* is used for information concerning use of this parameter during call. The heading *Return Code* is used for information concerning use of this parameter during return.

The names used for the parameters (FUNC, DATA_STRING, PARM_3, and PARM_4) in the preceding example are typical. The parameters can have any name, that meets these criteria:

- The names used to pass the first, third, and fourth parameters must be 2-byte (fullword) integer variables (PC/3270 DOS mode) or 4-byte (doubleword) long pointer variables that point to a 2-byte integer value.
- The name used to pass the second parameter must be a string variable.
- A unique name must be used for each parameter.

Since the same variable names are used on return as on call, it is not possible to place literal values for the parameters in the call statement. The call statement must contain four valid variable names. Parameter values are passed by reference. That is, the required value for each parameter must be stored in the associated variable name before the call statement is executed. Using the variable names, the call statement references the parameter values. The return value for each parameter is placed in the same variable name.

Because of differences in the way supported high-level languages handle data and because of different reserved words or syntax, source code requirements for calls to EHLLAPI functions vary for each supported language. Use of LIMs also affects the literal content of a specific call.

Specific call statement syntax for each supported high-level language is described in Chapter 5, “EHLLAPI Language Syntax and Considerations” on page 37.

Receiving Return Parameter Values

Before processing other returned parameters, your application should process the return code. The return code value is passed back to your application in the fourth variable name in the calling statement. If the return code value indicates successful execution of the function, continue processing the other parameters. If the return code value indicates something other than successful execution of the function, proceed to an error recovery routine.

PC/3270 and PC400 EHLLAPI return codes support a common error handling routine. See “Summary of Return Codes” on page 30 for more information.

Examples of return code processing are shown on pages 334, 341, 348, and 352.

EHLLAPI function calls return requested information as the following four parameters in this fixed order:

1. Function number
2. Data string
3. Length
4. Return code

Not all four parameters are changed on return from each function.

Function Number (first parameter) is always returned unchanged.

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

You need to preallocate space for returning data strings in your EHLLAPI application program. See “Memory Preallocation” on page 17.

Length (third parameter) is usually the length (in bytes) of the data string described in the immediately preceding paragraph. For certain functions, this parameter returns a presentation space position, an attribute byte, a field length, or a field position. The value associated with an attribute byte can be hex C0 to FF. The value for a PS position can be between 1 and 3564, decimal.

Return Code is usually a numeric return code. Your application should check the return code before continuing. If the return code indicates something other than successful execution of the function, your program should provide some means of recovery. PC/3270 and PC400 EHLLAPI Return codes allow for a common error-handling routine.

The return code parameter for the Convert Position or RowCol (99) function is a special case. The return code parameter for the Convert Position or RowCol (99) function returns data to the application program. If your EHLLAPI application calls

the Convert Position or RowCol (99) function, your application must provide for the processing of this data to avoid misleading or unpredictable results. Refer to "Convert Position or Convert RowCol (99)" on page 66 for more information.

Detailed explanations of return codes are contained in the descriptions of the individual EHLLAPI functions. A summary of return codes is provided under "Summary of Return Codes" on page 30.

Exiting Applications

Before terminating an application that uses EHLLAPI functions, you should execute a Reset System (21) function to ensure that:

The host is disconnected
Keystroke intercept is disabled
Host notification is disabled

Compiler/Assembler/Interpreter

Use a compiler, assembler, or interpreter from "Software Requirements" on page xx.

Linker

Except for applications written in the IBM Interpretive BASIC language, use the DOS linker provided with each supported high-level language. Linking is not required for IBM Interpretive BASIC.

Note: PC/3270 Windows mode and PC400 EHLLAPI do not support BASIC. For PC/3270 Windows mode or PC400 EHLLAPI, add at least one of these instructions into the **IMPORTS** item of the linker definition file:

PCSHLL.hllapi

PCSHLL.hllapiw

Program Execution

After you compile and link your application, you are ready to run it.

For the PC/3270 DOS mode, do as follows:

1. Before you type in the program filename and press Enter, you must:

   Have the memory required for execution of certain EHLLAPI functions preallocated. Refer to *Personal Communications Version 4.1 Up and Running for DOS (Full-Function Mode)* or *Personal Communications Version 4.1 Reference for DOS (Full-Function Mode)* for procedures to allocate required memory.

   Have the PC/3270 emulation program and EHLLAPI running.

2. Type:

   d:\path\yourprog

   where d: is the drive where your program resides, path is the directory path containing your program, and yourprog is the program file name.

3. Press the Enter key; your program begins.

For the PC/3270 Windows mode or the PC400, do as follows:

1. Select **RUN** in the file menu of the Program Manager.
2. Type:

   d:\path\yourprog

   where d: is the drive where your program resides, path is the directory path containing your program, and yourprog is the filename of your program.
3. Press the Enter key or click the **OK** button; your program begins.

Debugging

EHLLAPI supports debugging with a trace facility. This facility is enabled and disabled with the Set Session Parameters (9) function by setting the TRON or TROFF option. Refer to “Set Session Parameters (9)” on page 151 for more information concerning the Set Session Parameters (9) function.

When the trace facility is enabled, all requested functions are traced. When the trace facility is disabled, no requested functions are traced.

On call, the trace facility prints the function number of the requested function. On return, the trace facility prints the return code.

Note: This function is for the PC/3270 DOS mode EHLLAPI only. Both the PC/3270 Windows mode and PC400 EHLLAPI ignore it.

Programming Overview

Chapter 4. EHLLAPI Programming Guidelines

This chapter contains programming guidelines related to:

- Interaction of presentation space and EHLLAPI functions
- Common error-handling routines
- Interaction of separate EHLLAPI functions in common tasks
- ASCII mnemonics
- Diskette preparation (PC/3270 only)

Summary of Presentation Space Specification

Some functions interact with host presentation space. The supported specifications are discussed in this section.

Type of Presentation Space

A session can be declared as a display session or as a printer session during customization. EHLLAPI applications cannot connect to printer sessions.

PC/3270 and PC400 EHLLAPI support only the host system type of presentation space. If you are adapting an application prepared for another emulation program, see Appendix D, “DDE Sample Program.”

Size of Presentation Space

PC/3270 and PC400 EHLLAPI support:

- A minimum presentation space size of 1920 bytes, the number of bytes in a 24-row by 80-column screen.
- A maximum presentation space size of 3564 bytes, the number of bytes in a 27-row by 132-column screen.

You can specify a presentation space size between the two sizes; use the customization facility of PC/3270 or PC400.

To determine the size of a presentation space, examine bytes 12 and 14 of the returned data string for the Query Session Status (22) function. See “Query Session Status (22)” on page 118.

Presentation Space Names

PC/3270 DOS mode EHLLAPI supports up to eight presentation spaces or sessions. PC/3270 Windows mode and PC400 EHLLAPI support up to 26 presentation spaces or sessions. Valid names are the letters *A–H* (*A–Z* for PC/3270 Windows mode and PC400). To be valid within an application, a name must be declared during customization. See *Personal Communications Version 4.1 Up and Running for DOS (Full-Function Mode)*, *Personal Communications Version 4.1 Reference for DOS (Full-Function Mode)*, *Personal Communications Version 4.1 Up and Running for Windows*, *Personal Communications Version 4.1 Reference for Windows*, or *Personal Communications AS/400 and 3270 Version 4.1 Up and Running for Windows*.

© Copyright IBM Corp. 1996

No matter how many presentation space names are declared, EHLLAPI functions interact with only one presentation space at a time. The presentation space name (PSID) identifies the particular presentation space in which a function is to operate.

For some functions, the PSID is contained in a preceding call to the Connect Presentation Space (1) function. For other functions, the PSID is contained in the calling data string parameter.

Host-Connected Presentation Space

Connection to the host presentation space (or session) is controlled with the Connect Presentation Space (1) and Disconnect Presentation Space (2) functions. The connection status determines whether some functions can be executed. It also affects how the PSID is defined.

These guidelines explain how to control the status of the connection to the host presentation space:

At any given time, there can be either none or one host-connected presentation space.

Before the first connect is executed, there is no host-connected presentation space.

Following a connect, there is only one host-connected presentation space. The connected host presentation space is identified in the calling data string parameter of the connect function.

A subsequent call to connect can be executed with no intervening disconnect. In this case, there is still just one host-connected presentation space. Again, the connected host presentation space is identified in the calling data string parameter of the connect function.

Following a disconnect, there is no host-connected presentation space. This rule applies after multiple consecutive calls to connect or after a single call to connect.

You cannot connect to a logical printer session.

PSID Handling

The PSID is used to specify the host presentation space (or session) in which you desire a function to operate. The way the PSID is handled is affected by:

The method used to specify the PSID:

- As the calling data string parameter of a preceding call to the Connect Presentation Space (1) function
- As a character in the calling data string of the function being executed. Handling varies depending on whether the character is:
  - A letter *A*–*H* (*A*–*Z* for PC/3270 Windows mode and PC400)
  - A blank or a null

The status of the connection to the host presentation space.

The following paragraphs describe how the PSID is handled for the various combinations of these two factors.

PSID Handling for Functions Requiring Connect

Some functions interact only with the host-connected presentation space. These functions require the Connect Presentation Space (1) function as a prerequisite call. See “Summary of Prerequisite Calls” on page 28. The PSID for these functions is determined by the Connect Presentation Space (1) and the Disconnect Presentation Space (2) functions as follows:

When there is no host-connected presentation space, these functions do not interact with a presentation space. A return code of 1 is generated.

When there is one host-connected presentation space, these functions interact with the presentation space specified in the calling data string parameter of the most recent call to the Connect Presentation Space (1) function.

PSID Handling for Functions Not Requiring Connect

Some functions can interact with a host presentation space whether it is connected or not. These functions allow you to specify the PSID in the calling data string parameter:

Connect Presentation Space (1)

Convert Position RowCol (99)

Get Key (51)

Post Intercept Status (52)

Query Close Intercept (42)

Query Host Update (24)

Query Session Status (22)

Query Sessions (10)

Query System (20)

Start Close Intercept (41)

Start Host Notification (23)

Start Keystroke Intercept (50)

Stop Close Intercept (43)

Stop Host Notification (25)

Stop Keystroke Intercept (53)

All except the first two of these functions allow you to specify the PSID using either:

A letter *A*–*H* (*A*–*Z* for PC/3270 Windows mode and PC400)
A blank or a null

The first two functions require that a letter be used to specify PSID.

When there is no host-connected presentation space, the following rules apply:

1. The function can interact with any declared host presentation space if a letter, not a blank or a null, specifies the PSID.
2. If a blank or a null is used to specify PSID, a return code of 1 is generated. The function does not execute.

3. Using a letter to specify PSID does not establish a host-connected presentation space. Unless a connect is executed:

   Functions requiring a connect do not interact with any presentation space.

   Functions not requiring a connect do not work if a blank or a null is used to specify PSID.

When there is one host-connected presentation space, the following rules apply:

1. The function can interact with any declared host presentation space if a letter specifies the PSID.
2. If a blank or a null is used to specify PSID, the function operates in the presentation space identified in the most recent call to the Connect Presentation Space (1) function.
3. Using a letter to specify PSID does not change the established PSID of the host-connected presentation space. Unless another connect is executed:

   Functions requiring a connect continue to interact with the PSID specified in the most recent call to connect.

   If a blank or a null is used to specify PSID, functions not requiring a connect also still interact with the PSID specified in the most recent call to connect.

Summary of Prerequisite Calls

Table 2 lists all prerequisite call requirements for each EHLLAPI function. These requirements apply anytime you call an EHLLAPI function either when running an application program or when using the Program Sampler described in Chapter 6, "EHLLAPI Functions."

Table 2 (Page 1 of 2). Summary of Prerequisite Calls for the EHLLAPI Functions

Function	Prerequisite Calls
Connect Presentation Space (1)	None
Disconnect Presentation Space (2)	Connect Presentation Space (1)
Send Key (3)	Connect Presentation Space (1)
Wait (4)	Connect Presentation Space (1)
Copy Presentation Space (5)	Connect Presentation Space (1)
Search Presentation Space (6)	Connect Presentation Space (1)
Query Cursor Location (7)	Connect Presentation Space (1)
Copy Presentation Space to String (8)	Connect Presentation Space (1)
Set Session Parameters (9)	None
Query Sessions (10)	None
Reserve (11)	Connect Presentation Space (1)
Release (12)	Connect Presentation Space (1)
Copy OIA (13)	Connect Presentation Space (1)
Query Field Attribute (14)	Connect Presentation Space (1)
Copy String to Presentation Space (15)	Connect Presentation Space (1)
Storage Manager (17)	None
Pause (18)	None
Query System (20)	None

Table 2 (Page 2 of 2). Summary of Prerequisite Calls for the EHLLAPI Functions

Function	Prerequisite Calls
Reset System (21)	None
Query Session Status (22)	None
Start Host Notification (23)	None
Query Host Update (24)	Start Host Notification (23)
Stop Host Notification (25)	Start Host Notification (23)
Search Field (30)	Connect Presentation Space (1)
Find Field Position (31)	Connect Presentation Space (1)
Find Field Length (32)	Connect Presentation Space (1)
Copy String to Field (33)	Connect Presentation Space (1)
Copy Field to String (34)	Connect Presentation Space (1)
Set Cursor (40)	Connect Presentation Space (1)
Start Close Intercept (41)	None
Query Close Intercept (42)	Start Close Intercept (41)
Stop Close Intercept (43)	Start Close Intercept (41)
Start Keystroke Intercept (50)	None
Get Key (51)	Start Keystroke Intercept (50)
Post Intercept Status (52)	Start Keystroke Intercept (50)
Stop Keystroke Intercept (53)	Start Keystroke Intercept (50)
Send File (90)	None
Receive File (91)	None
Convert Position or RowCol (99)	None
Connect Window Services (101)	None
Disconnect Window Service (102)	Connect Window Services (101)
Query Window Coordinates (103)	Connect Window Services (101)
Window Status (104)	Connect Window Services (101)
Change Switch List LT Name (105)	Connect Window Services (101)
Change PS Window Name (106)	Connect Window Services (101)
Start Playing Macro (110)	Connect Presentation Space (1)
Connect for Structured Fields (120)	None
Disconnect from Structured Fields (121)	Connect for Structured Fields (120)
Query Communications Buffer Size (122)	None
Allocate Communications Buffer (123)	None
Free Communications Buffer (124)	Allocate Communications Buffer (123)
Get Request Completion (125)	Connect for Structured Fields (120) and Allocate Communications Buffer (123) and either Read Structured Fields (126) or Write Structured Fields (127)
Read Structured Fields (126)	Connect for Structured Fields (120) and Allocate Communications Buffer (123)
Write Structured Fields (127)	Connect for Structured Fields (120) and Allocate Communications Buffer (123)

Summary of Return Codes

Return codes are explained in detail in the descriptions of the individual functions. Generally, the meaning of a return code is consistent with the standard return codes discussed in Table 3 This consistency is maintained so that you can use a common error-handling routine in your EHLLAPI program.

Your EHLLAPI 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 EHLLAPI program should call a subroutine that requests the Query System (20) function and exit. The subroutine could extract extended error-code information to help your service coordinator determine the cause of the system error. When you contact your service coordinator, be prepared to give the system-error information generated by the Query System (20) function.

A subset of the standard return codes can apply to all functions in a general way. When used with individual functions, the return codes can have slightly different interpretations. Refer to the individual functions for the best explanations of that function's return codes.

Table 3. Explanations of Return Codes

Return Code	Explanation
0	Function successfully executed or zero updates since the last call was issued.
1	An incorrect host presentation space ID was specified. Specified session either was not connected, does not exist, or is a logical printer session.
2	A parameter error was encountered or an incorrect function code was specified. (Refer to individual function for details.)
4	The execution of the function was inhibited because the target presentation space was busy, in X CLOCK state (X []), or in X SYSTEM state.
5	The execution of the function was inhibited for some reason other than those stated in return code 4.
6	A data error was encountered due to specification of an incorrect parameter (for example, a length error causing truncation).
7	The specified presentation space position was incorrect.
8	A functional procedure error was encountered; for example, use of conflicting functions or missing prerequisite functions.
9	A system error was encountered. Whenever you receive a return code 9, you should issue a call to the Query System (20) function and exit. Extended error codes 1 and 2 that are returned by this function are vital in helping your service coordinator determine the cause of the system error. When you contact your service coordinator, be prepared to provide the system error information generated by the Query System (20) function.
10	This function is not available for EHLLAPI.
11	This resource is not available. (PC/3270 Windows mode and PC400)
12	This session stopped. (PC/3270 Windows mode and PC400)
24	String not found or unformatted presentation space.

Considerations for Using Send File (90) and Receive File (91) Functions (PC/3270 DOS Mode Only)

The way DOS allocates memory and searches for .EXE files affects the use of Send File (90) and Receive File (91) functions. The following paragraphs describe how your application can be affected.

Using the DOS SETBLOCK Command

The memory requirements of the Send File (90) and Receive File (91) functions occasionally require use of the DOS SETBLOCK command. If an insufficient memory error occurs while your application is executing either the Send File (90) function or the Receive File (91) function, add a DOS SETBLOCK call to allocate the memory required to execute these functions.

Note: For the /2 languages, the DOS SETBLOCK command is not required.

Functions Send File (90) and Receive File (91) can operate only if there is enough free memory for the appropriate program to be loaded. If insufficient space is available, a return code of 308 is generated.

In general, Interpretive BASIC and C do not occupy all of memory, but either can still cause a 308 return code to be generated. 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.EXE (which contains the loader).

A well-behaved program uses the DOS SETBLOCK function call when it receives control from COMMAND.EXE 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).

The MAX ALLOC and the MIN ALLOC field parameters in the EXE file header also affect the memory size.

If left unchanged, compiled BASIC, COBOL, and Pascal programs are given all of the available memory in the DOS session when linked with their LIMs and loaded. Since there is no uniform way for PC/3270 EHLLAPI to perform a SETBLOCK for all supported languages, it does not do it. To free memory required by functions Send File (90) and Receive File (91), the DOS SETBLOCK function (DOS interrupt X'21' AH=X'4A') must be implemented within your application as required by the selected high-level language.

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

System File Preparation

PC/3270 EHLLAPI application programs and the data they process can be arranged in numerous ways in directories and subdirectories on one diskette, on one hard disk, or on various combinations of the two. Depending upon how you arrange your program and data, the following considerations might apply to you.

SEND.EXE and RECEIVE.EXE Files

If your application calls the Send File (90) function or the Receive File (91) function, you must observe the following guidelines to ensure that the related executable (.EXE) file is available.

When either of these functions is called, the drive is specified in the calling PS Position parameter. DOS looks only in the current directory of the specified drive for the related .EXE file. If the file is not in *that* directory, an error occurs.

Each of these approaches prevents this error:

1. Store the .EXE files in a directory that is declared as current. Do not change the current directory.
2. If your application requires that the current directory be changed, do one of the following steps:

   Change the current directory to the directory in which the .EXE file is stored before calling the function.

   Store the .EXE file on a drive where you do not change the current directory and always specify that drive in the calling PS Position parameter.

   Store the .EXE file in every directory that can be current when the related function is called.

Note: If your EHLLAPI application is to be run on a personal computer without a hard disk, a diskette-swapping prompt might be required to ensure that the needed .EXE file is available.

ASCII Mnemonics

Keystrokes originating at a host keyboard might have a corresponding ASCII value. The response of the Get Key (51) function to a keystroke depends on: whether the key is defined and also on whether the key is defined as an ASCII value or an ASCII mnemonic.

The keyboard for one session might be incapable of producing some codes needed by another session. ASCII mnemonics representing these codes can be included in the data string parameter of the Send Key (3) function.

The capabilities of the Send Key (3) and Get Key (51) functions allow sessions to exchange keystrokes that cannot be represented by ASCII values or by an available key. A set of mnemonics that can be generated from a keyboard is provided. These mnemonics let you use ASCII characters to represent the special function keys of the personal computer keyboard.

Mnemonics for unshifted keys consist of the escape character followed by an abbreviation. In PC/3270 DOS mode, this is also true for the shift keys themselves, shift, Alt and Ctrl. Mnemonics for shifted keys consist of the mnemonic for the shift key followed by the mnemonic for the unshifted key. Hence, the mnemonic for a shifted key is a 4-character sequence: escape character, abbreviation, escape character, abbreviation.

The default escape character is @. The value of the escape character can be changed to any other character with the ESC=c option of the Set Session Parameters (9) function. The following text uses the default escape character.

Shift indicators that are not part of the ASCII character set are represented to the host application by these 2-byte ASCII mnemonics:

Shift	@S
Alt	@A
Ctrl	@r

Mnemonics for these shift indicators are never received separately by an application. Likewise, they are never sent separately by an application. Shift indicator mnemonics are always accompanied by a non-shift-indicator character or mnemonic.

Note: PC/3270 Windows mode and PC400 cannot treat shift indicators (@S, @A, and @r).

The abbreviations used make the mnemonics for special keys easy to remember. An alphabetic key code has been used for the most common keys. For example, the Clear key is *C*; the Tab key is *T*, and so on. Please note that the uppercase and lowercase alphabetic characters are mnemonic abbreviations for different keys.

The following text describes the use of these functions.

General

All defined keys are represented by either:

A 1-byte ASCII value that is part of the 256-element ASCII character set

A 2-, 4- or 6-byte ASCII mnemonic

To represent a key defined as an ASCII character, a 1-byte ASCII value corresponding to that character is used.

To represent a key defined as a function, a 2-, 4- or 6-byte ASCII mnemonic that corresponds to that function is used. For example, to represent the backtab key, @B is used. To represent F1, @1 is used. To represent Erase Input, @A@F is used. See the following tables.

@B	Backtab	@N	New Line	@s	Scroll Lock
@C	Clear*	@O	Space (PC400)	@t	Num Lock
@D	Delete	@P	Print	@u	Page Up (PC400)
@E	Enter*	@R	Reset	@v	Page Down (PC400)
@F	Erase EOF	@T	Tab	@@	@ (at) symbol
@H	Help	@U	Cursor Up	@$	Alternate Cursor
@I	Insert	@V	Cursor Down	@<	Backspace
@J	Jump	@Y	Caps Lock		
@L	Cursor Left	@Z	Cursor Right		

Note: The @Y, @s and @t mnemonics can be intercepted by the Get Key (51) function, but they cannot be sent by the Send Key (3) function.

@0	Home	@7	PF7*	@e	PF14*	@l	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* (PC/3270)
@5	PF5*	@c	PF12*	@j	PF19*	@y	PA2* (PC/3270)
@6	PF6*	@d	PF13*	@k	PF20*	@z	PA3* (PC/3270)

* An AID key mnemonic.

@A@C	Test* (PC400)	@A@o	Jump to Session 2 (PC400)
@A@D	Word Delete (PC/3270)	@A@p	Jump to Session 3 (PC400)
@A@E	Field Exit (PC400)	@A@q	Jump to Session 4 (PC400)
@A@F	Erase Input	@A@r	Jump to Session 5 (PC400)
@A@H	System Request*	@A@t	Print (Personal Computer) (PC400)
@A@I	Change Insert Mode (PC400)	@A@y	Forward Word Tab
@A@J	Cursor Select*	@A@z	Backward Word Tab
@A@L	Cursor Left Fast	@A@-	Field- (PC400)
@A@Q	Attention*	@A@+	Field+ (PC400)
@A@R	Device Cancel	@A@<	Backward Record* (PC400)
@A@T	Print Screen	@G@a	Jump to Session A (PC/3270)
@A@U	Cursor Up Fast (PC400)	@G@b	Jump to Session B (PC/3270)
@A@V	Cursor Down Fast (PC400)	@G@c	Jump to Session C (PC/3270)
@A@Z	Cursor Right Fast	@G@d	Jump to Session D (PC/3270)
@A@9	Reverse Video	@G@e	Jump to Session E (PC/3270)
@A@a	Backspace Erase (PC/3270)	@G@f	Jump to Session F (PC/3270)
@A@b	Underscore	@G@g	Jump to Session G (PC/3270)
@A@c	Reset Receive Video	@G@h	Jump to Session H (PC/3270)
@A@d	Red	@G@s	Jump to DOS Session (PC/3270)
@A@e	Pink	@S@x	Dup
@A@f	Green	@S@y	Field Mark (PC/3270)
@A@g	Yellow	@X@c	Split vertical bar (¦) (PC400)
@A@h	Blue	@X@1	SO/SI Display (PC400)
@A@i	Turquoise	@X@5	SO/SI Generate (PC400)
@A@j	White	@X@6	Attribute Display (PC400)
@A@l	Reset Host Colors	@X@7	Character Forward (PC400)
@A@n	Jump to Session 1 (PC400)		

Notes:

1. The first @ symbol in the first two tables represents the escape character. The first and second @ symbol in the remaining two tables is the escape character. The @ symbol is the default escape character. The value of the escape character can be changed using the ESC=c option of the Set Session Parameters (9) function.

   If the escape character is changed to #, the literal sequences used to represent the Backtab, Home, and Erase Input keys become #B, # , and #A#F, respectively. The literal sequence used to represent the @ symbol becomes #@.

2. If you send a print screen mnemonic (@P or @A@T), place the mnemonic at the end of the calling data string.

3. If you send the mnemonic for device cancel (@A@R), the mnemonic is passed through with no error message. However, a local copy is not stopped.

Get Key (51) Function

If the operator types a key defined as an ASCII character, the host application receives a 1-byte ASCII value that corresponds to that character.

If the operator types a key defined as a function, the host application receives a 2-, 4- or 6-byte ASCII mnemonic that corresponds to that function. If the backtab key is typed, @B is received. If F1 is pressed, @1 is received. If Erase Input is pressed, @A@F is received.

If the operator types a defined shift key combination, the host application receives the ASCII character or the 2-, 4-, or 6-byte ASCII mnemonic that corresponds to the defined character or function. For example, if the combination Ctrl-p is defined as reset, @R is received when Ctrl-p is typed.

If the operator types an individual undefined key, the Get Key (51) function returns a return code of 20. Nothing is sent to the host application.

If the operator types an undefined shift key combination, the result depends on whether the unshifted state of the key is defined:

If the unshifted state is defined as an ASCII character, the host application receives the ASCII mnemonic for the shift key followed by the character associated with the unshifted key. For example, if the Alt key is pressed in combination with a key defined as lowercase a, the host application receives @Aa (PC/3270 DOS mode only).

If the unshifted state is defined as a function, the host application receives the ASCII mnemonic for the shift key followed by the ASCII mnemonic that corresponds to the defined function. For example, suppose the p key is mapped to be reset and that Ctrl-p is undefined. If Ctrl-p is pressed, the host application receives @r@R (the mnemonics for Ctrl-reset), not @rp (the mnemonics for Ctrl-p) (PC/3270 DOS mode only).

If the unshifted state is undefined, the Get Key (51) function returns a return code of 20. The host application receives nothing.

The Get Key (51) function precedes all characters and mnemonics sent to the host application with two ASCII characters as described in the preceding text. The first ASCII character is the PSID of the host presentation space to which the keystrokes are sent. The second character is A, S, or M for ASCII, special shift, or mnemonic, respectively. See page 102. For undefined shift key combinations, the second character is always S.

Note: PC/3270 Windows mode and PC400 do not return S as a second character.

Send Key (3) Function

To send an ASCII character to another session, include that character in the data string parameter of the Send Key (3) function.

To send a function key to another session, include the ASCII mnemonic for that function in the data string parameter of the Send Key (3) function.

If the Send Key (3) function sends an unrecognized mnemonic to the host session, @AA for example, a return code rejecting the key might result.

EHLLAPI Programming Guidelines

Chapter 5. EHLLAPI Language Syntax and Considerations

This chapter provides information needed to call EHLLAPI functions in each of the supported high-level languages. Source code syntax, compiler commands and options, and linker commands and options are discussed for each supported high-level language. Refer to the section named for the high-level language in which you are programming for more information.

For the PC/3270 DOS mode EHLLAPI, refer to:

"IBM BASIC (Compiler) Guidelines" on page 38
"IBM BASIC (Interpretive) Guidelines" on page 39
"IBM C Guidelines" on page 41
"IBM COBOL Guidelines" on page 42
"IBM Macro Assembler Guidelines" on page 43
"IBM Pascal Guidelines" on page 45

For the PC/3270 Windows mode and PC400 EHLLAPI, refer to:

"C for Windows Guidelines" on page 48
"Macro Assembler for Windows Guidelines" on page 49

© Copyright IBM Corp. 1996

IBM BASIC (Compiler) Guidelines

The following guidelines are necessary in preparing EHLLAPI applications that use EHLLAPI functions in the Compiler BASIC language.

Source Code Preparation

Specific statements that you must include in Compiler BASIC source code to use EHLLAPI functions are listed in the following text. Also, the syntax for calls to specific EHLLAPI functions is shown.

Required Statements

There are no required statements for Compiler BASIC applications that call EHLLAPI functions.

EHLLAPI Function Call Syntax

The syntax for calls to EHLLAPI functions in Compiler BASIC can be either of these:

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

Note: Some PC/3270 EHLLAPI functions require preallocation of memory for the return data string parameter. See “Memory Preallocation” on page 17.

Compiling

Compile your PC/3270 EHLLAPI application as instructed in the *IBM Compiler BASIC* manual.

Linking

Link your PC/3270 EHLLAPI application as follows:

1. Have your compiled PC/3270 EHLLAPI application program and the file HLLCBAS.OBJ from your PC/3270 diskette available. The drive location and path information of these programs will have to be added to the link command in step 2.
2. Type the following command and press Enter at the DOS prompt:

   LINK d:\path\yourprog+d:\path\HLLCBAS

   where d:\path\ is the drive and path specification for the associated file and yourprog is the name of the file containing the compiled object code.

Considerations for using IBM BASIC (Compiler)

Compiled BASIC programs are often developed and debugged as interpretive BASIC programs. In the case of EHLLAPI applications, the Copy Presentation Space (5) function is not available in interpretive BASIC. Except for calls to the Copy Presentation Space (5) function, applications can be tested in interpretive BASIC and then compiled. The Copy Presentation Space (5) function syntax and logic need to be verified in the compiled version.

Also, interpretive BASIC applications require initialization statements that compiled BASIC applications do not require. These statements are compatible with compiled BASIC. They can be removed to save memory if desired.

IBM BASIC (Interpretive) Guidelines

The following guidelines are helpful in preparing EHLLAPI applications that use EHLLAPI functions in the Interpretive BASIC language.

Source Code Preparation

Specific statements that you must include in Interpretive BASIC source code to use EHLLAPI functions are listed in the following text. Also, the syntax for calls to specific EHLLAPI functions is shown.

Required Statements

The following statements are required for QBASIC applications that call EHLLAPI functions. These statements must be executed before the first call to an EHLLAPI function is executed.

```
DEFINT A-Z: DEF SEG= : APILOC=PEEK(&H1FE)+PEEK(&H1FF)
IF APILOC=  THEN PRINT "EHLLAPI NOT AVAILABLE": END
DIM APICALL(73): FOR I=  TO 72: READ APICALL(I): NEXT
DATA &H8B55, &H1EEC, &H8B 6, &H C76, &H 48B, &H 53C, &H D75
DATA &H768B, &HC7 6, &H 2 4, &H 7  , &H5D1F, &H 8CA, &H3C
DATA &H7511, &HB8 E, &H 1 4, &H768B, &H8A 6, &H 73C, &H5D1F
DATA &H 8CA, &H2E  , &H85A2, &H8B  , &H A76, &H 48A, &H  B4
DATA &HA32E, &H  8F, &H448B, &H2E 2, &H88A3, &H8C  , &H2ED8
DATA &H86A3, &H8B  , &H 876, &H 48B, &HA32E, &H  8A, &H768B
DATA &H8B 6, &H2E 4, &H8DA3, &H8D  , &H8236, &H8C  , &H8EC8
DATA &HB8D8, &H 1 4, &H  BB, &HCD  , &H8B7F, &H 676, &H892E
DATA &H2E1C, &H1E8B, &H  8A, &H768B, &H2E 8, &H1C89, &H1F 7
DATA &HCA5D, &H   8, &H435 , &H  42, &H    , &H    , &H
DATA &H    , &H    , &H
```

EHLLAPI Function Call Syntax

The syntax for calls to EHLLAPI functions in QBASIC is as follows:

```
DEF SEG = VARSEG(APICALL( ): BLIM=VARPTR(APICALL( ))
CALL ABSOLUTE(FUNC%, SDATA$, DLEN%, RETC%, BLIM)
```

Note: Some PC/3270 EHLLAPI functions require preallocation of memory for the return data string parameter. See “Memory Preallocation” on page 17.

Compiling

Not applicable for Interpretive BASIC.

Linking

Not applicable for Interpretive BASIC.

Considerations for Using IBM BASIC (Interpretive)

Users of BASIC have a choice of two programming environments: Interpretive BASIC and Compiled BASIC.

Copy Presentation Space (5) Function and BASIC

The **Copy Presentation Space** (5) function is not available to application programs written in Interpretive BASIC.

Required Statements for Interpretive BASIC

To access the Interpretive BASIC LIM that is in the EHLLAPI module, applications must contain special initialization statements. These statements are shown under “Required Statements” on page 39. These statements locate the LIM from a fixed pointer location and then set the SEG and OFFSET values for the subroutine call. If you do not understand the routine involved, do not worry; just copy the statements shown into your program's initialization routine.

Note: While these statements are not required by Compiled BASIC, they are compatible with it. This allows you to develop a program using Interpretive BASIC and then to compile it for improved performance.

String Processing in BASIC

BASIC manages memory space for strings dynamically. That is, as you assign new data to a string variable, BASIC locates the data at a new place in memory. This aspect of BASIC has important implications for PC/3270 EHLLAPI applications.

If you write an application that uses strings extensively, lengthy pauses can occur while BASIC *packs* memory.

Dynamic Storage Pool

BASIC's method of memory management for string variables can affect performance. If your application makes extensive use of string manipulations, you should take steps to avoid degraded performance.

BASIC's dynamic memory management scheme involves assigning a new location to a variable each time a new value is calculated for it. The first string is located at one end of the available memory. Each new assignment is closer to the other end of memory. Eventually, the other end of memory will be reached. Before the next variable can be processed, BASIC must free up memory by moving all string variables toward the starting end of memory. While this is being done, your computer will appear to be locked up. This process can last several minutes.

By using the proper programming techniques and BASIC statements to calculate string values, you might be able to reduce or eliminate these delays. For more information, refer to *IBM PC BASIC Manual*.

IBM C Guidelines

The following guidelines are helpful in preparing EHLLAPI applications that call EHLLAPI functions in the IBM C language.

Source Code Preparation

Specific statements that you must include in IBM C source code to call EHLLAPI functions are listed in the following text. Also, the syntax for calls to specific EHLLAPI functions is shown.

Required Statements

The IBM C language requires a variable type declaration statement in your application as follows: int FUNC, STRLEN, RETC; char DATA_STRING [size]; *OR*
char * DATA_STRING;

EHLLAPI Function Call Syntax

The syntax for calls to EHLLAPI functions in IBM C can be either:

```
hllc(&FUNC,DATA_STRING,&STRLEN,&RETC);

clim(&FUNC,DATA_STRING,&STRLEN,&RETC);
```

Note: Some PC/3270 EHLLAPI functions require preallocation of memory for the return data string parameter. See "Memory Preallocation" on page 17.

Compiling

Compile your PC/3270 EHLLAPI application as instructed in *IBM Personal Computer Series: C Compiler Compile, Link and Run*.

Linking

Link your PC/3270 EHLLAPI application as follows:

1. Have your compiled PC/3270 EHLLAPI application program and the file HLLC_x.OBJ from your PC/3270 diskette available (where x can be C, S, M, or L depending on the memory model required by your application). The drive location and path information of these programs will have to be added to the link command in step 2.
2. Depending on whether you are using IBM C or IBM C/2, type one of the following commands and press Enter at the DOS prompt.

   For IBM C version 1.0, type:

   ```
   CLINK d:\path\yourprog+d:\path\HLLC_x
   ```

   For IBM C/2 version 1.0, type:

   ```
   LINK d:\path\yourprog+d:\path\HLLC_x
   ```

   where d:\path\ is the drive and path specification for the associated file, yourprog is the name of the file containing the compiled object code, and x can be C, S, M or L depending on the memory model required by your application.

   This produces an executable module called YOURPROG.EXE. After loading PC/3270 EHLLAPI, you can execute this IBM C program.

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

IBM COBOL Guidelines

The following guidelines are helpful in preparing EHLLAPI applications that call EHLLAPI functions in the COBOL language.

Source Code Preparation

Specific statements that you must include in COBOL source code to call EHLLAPI functions are listed in the following text. Also, the syntax for calls to specific EHLLAPI functions is shown.

Required Statements

To incorporate calls to EHLLAPI functions in IBM COBOL applications, place the following statements in the Data Division.

```
 1   FUNC                PIC 9( 4)        USAGE COMP- .
 1   RETC                PIC 9( 4)        USAGE COMP- .
 1   STRLEN              PIC 9( 4)        USAGE COMP- .
O1   DATA_STRING            PIC X(size)      VALUE SPACES.
```

EHLLAPI Function Call Syntax

The syntax for calls to EHLLAPI functions in COBOL can be either:

```
CALL 'HLLCOB' USING FUNC  DATA_STRING  STRLEN  RETC.

CALL 'COBLIM' USING FUNC  DATA_STRING  STRLEN  RETC.
```

Note: Some PC/3270 EHLLAPI functions require preallocation of memory for the return data string parameter. See “Memory Preallocation” on page 17.

Compiling

Compile your PC/3270 EHLLAPI application with the command:

```
COBOL yourprog    /LITLINK;
```

where yourprog is the file name of your program.

Linking

Link your PC/3270 EHLLAPI application as follows:

1. Have your compiled PC/3270 EHLLAPI application program and the file HLLCOB.OBJ from your PC/3270 diskette available. The drive location and path information of these programs will have to be added to the link command in step 2.
2. Type the following command and press Enter at the DOS prompt:

   ```
   LINK d:\path\yourprog+d:\path\HLLCOB
   ```

   where d:\path\ is the drive and path specification for the associated file and yourprog is the name of the file containing the compiled object code.

   This will produce an executable module called YOURPROG.EXE. After loading PC/3270 EHLLAPI, you can execute this program in the normal manner.

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

IBM Macro Assembler Guidelines

The following guidelines are helpful in preparing EHLLAPI applications that call EHLLAPI functions in the IBM Macro Assembler language.

Source Code Preparation

Specific statements that you must include in IBM Macro Assembler source code to call EHLLAPI functions are in the following text. Also, the syntax for calls to EHLLAPI functions is shown.

Except for the Storage Manager (17) function, calls to EHLLAPI functions require a Parameter Control Block (PCB). The Storage Manager (17) function uses a register interface. Therefore, the Storage Manager (17) function is discussed separately in the following paragraphs.

Save any required non–segment registers before issuing the hex 7F interrupt.

After the requested function is executed, control is returned to the next sequential instruction after the hex 7F interrupt.

Except for the Storage Manager (17) function, the return code parameter and any result codes are passed back in the PCB.

Note: Note that the data string parameter is passed back directly. You must retain the user's data segment and offset addresses.

Results of the Storage Manager (17) function are stored in the registers.

Required Statements, General

Except for the Storage Manager (17) function, calls to EHLLAPI functions from an application written in the IBM Macro Assembler language require a PCB like the one shown below, and statements that set the following registers as shown:

DS:SI point to the PCB.

AH 01

AL 04

BX 00

The general format for a PCB is as follows:

```
PCB_HDR     DB   'PCB'  ;PCB Header (in Caps)
PCB_FUNC    DB   0        ;function code parameter VALUE
PCB_DSEG    DW   0        ;String segment address
PCB_DADDR   DW   0        ;String offset address
PCB_STRLEN  DW   0        ;length parameter VALUE
PCB_FILLER  DB   0        ;Not used
PCB_RETC    DW   0        ;return code parameter VALUE
PCB_MAXLEN  DW   25000    ;Maximum user string size
```

Observe the following guidelines when writing a PCB:

The control block header PCB must be in uppercase and must appear at the start of the control block. See PCB_HDR.

The function code parameter value must be a 1-byte number in binary format that is the function code value of the desired EHLLAPI function. See PCB_FUNC.

Two entries in the PCB are required to point to the user's data string. This data must point to the actual location in memory where the string resides, and not to a length or pointer prefix.

PCB_SEG contains the segment address. This entry should always precede the offset address.

PCB_DADDR contains the offset address. This entry should always follow the segment address.

PCB_STRLEN is the length parameter in binary format.

PCB_RETC is the fourth parameter, which is often a PS position when calling an EHLLAPI function.

For the IBM Macro Assembler Language, set PCB_MAXLEN to 25000.

Required Statements, Storage Manager (17) Function

The Storage Manager (17) function, unlike all the other EHLLAPI functions, uses a register interface. The other EHLLAPI functions use a PCB interface that is described in the preceding paragraph. The required statements for the Storage Manager (17) function set the following registers as required by the specific subfunction. To complete execution of the called subfunction, examine the contents of the pertinent registers. The following list explains the contents of each register on call and return.

Register	Explanation
On **Call**	
AX	0104H (mandatory)
BX	rrAAH where rr is the request code. Valid input is binary. 01 for Get Storage subfunction 02 for Free Storage subfunction 03 for Query Free Storage subfunction 04 for Free All Storage subfunction
CX	Storage size for Get Storage request, or storage block ID for Free Storage request
On **Return**	
AX	Storage segment address for Get Storage request
BX	Storage offset address for Get Storage request
CX	Storage Block ID for Get Storage request
DX	Return Code value for all three requests

EHLLAPI Function Call Syntax

The syntax for calls to EHLLAPI functions in IBM Macro Assembler is:

INT 7FH

Note: Some PC/3270 EHLLAPI functions require preallocation of memory for the return data string parameter. See "Memory Preallocation" on page 17.

Assembling

For IBM Macro Assembler applications that call to EHLLAPI functions, assemble the source code as instructed in *IBM Macro Assembler*.

Linking

Link your assembled code with this command:

LINK d:\path\yourprog

where d:\path\ is the drive and path specification for the associated file and yourprog is the name of the file containing the compiled object code.

This will produce an executable module called YOURPROG.EXE. After loading PC/3270 EHLLAPI, you can execute this program in the normal manner.

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

IBM Pascal Guidelines

The following guidelines are helpful in preparing EHLLAPI programs that call EHLLAPI functions in the IBM Pascal language.

Source Code Preparation

Specific statements that you must include in IBM Pascal source code to use calls are listed in the following text. Also, the syntax for calls to specific EHLLAPI functions is shown.

Required Statements

The following statements are required to incorporate calls to EHLLAPI functions in IBM Pascal applications.

```
VAR
 FUNC,       (* function code parameter *)
 STRLEN,     (* length parameter          *)
 RETC : INTEGER;       (* return code parameter    *)
 DATA_STRING : STRING(size);    (* data string parameter     *)

PROCEDURE HLLPAS(     VAR FUNC : INTEGER;
       VAR DATA_STRING : STRING;
       VAR  STRLEN,
              RETC : INTEGER); EXTERN;
```

EHLLAPI Function Call Syntax

The syntax for calls to EHLLAPI functions in IBM Pascal can be either:

```
HLLPAS(FUNC, DATA_STRING, STRLEN, RETC);

PASLIM(FUNC, DATA_STRING, STRLEN, RETC);
```

Note: Some PC/3270 EHLLAPI functions require preallocation of memory for the return data string parameter. See “Memory Preallocation” on page 17.

Compiling

Compile your EHLLAPI program as instructed in *IBM Personal Computer Language Series: Pascal*.

Linking

Link your PC/3270 EHLLAPI application as follows:

1. Have your compiled PC/3270 EHLLAPI application program and the file HLLPAS.OBJ from your PC/3270 diskette available. The drive location and path information of these programs will have to be added to the link command in step 2.
2. Type the following command and press Enter at the DOS prompt:

   LINK d:\path\yourprog+d:\path\HLLPAS

   where d:\path\ is the drive and path specification for the associated file and yourprog is the name of the file containing the compiled object code.

   This produces an executable module called YOURPROG.EXE. After loading PC/3270 EHLLAPI, you can execute this program in the normal manner.

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

Considerations for Using Pascal

Several EHLLAPI functions transfer information in the data string parameter, either on call or return, in a series of fields. Although IBM Pascal provides no explicit substringing functions for variables of type STRING, there are ways to work with the fields within the Data String parameter.

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

```
PROGRAM QSYS (INPUT,OUTPUT);

TYPE
                                        ( Query System overlay type      )
                                        ( [xx] necessary for proper      )
    QSYSOVLTYP = RECORD                 ( byte alignment                 )
     HLLVERS [  ]: STRING(1);     ( EHLLAPI version number          )
     HLLLVL    [ 1]: STRING(2);     ( EHLLAPI level number            )
     HLLDATE [ 3]: STRING(6);     ( EHLLAPI date                    )
     RESERV1 [ 9]: STRING(3);     ( Reserved                        )
     HARDW      [12]: STRING(1);     ( Hardware base                   )
     PTYPE      [13]: STRING(1);     ( Program type                    )
     RESERV2 [14]: STRING(2);     ( Reserved                        )
     PVERS      [16]: STRING(2);     ( Program version number          )
     RESERV3 [18]: STRING(1);     ( Reserved                        )
     COMPRC    [19]: STRING(4);     ( Component error return code )
     SYSRC      [23]: STRING(4);     ( System error return code       )
     MODTYP    [27]: STRING(2);     ( Model type                      )
     RESERV4 [29]: STRING(2);     ( Reserved                        )
     MONTYP    [31]: STRING(1);     ( Monitor type                    )
     RESERV5 [32]: STRING(3);     ( Reserved                        )
    END;

VAR
 FUNC,                                  ( Function number                )
 LEN,                                          ( Length of the data string )
 RETC :INTEGER;                         ( EHLLAPI return code            )
 Q_SYS_OVL: QSYSOVLTYP;                 ( Query System Overlay           )
 Q_SYS_STR: STRING(35);                 ( Query System data string     )
 Q_SYS_PTR: ADS OF QSYSOVLTYP;   ( Pointer to the Q System       )
                                        ( Overlay                        )


PROCEDURE HLLPAS (VAR LIM_FUNC: INTEGER;
                  VAR  LIM_STR:    STRING;
                  VAR  LIM_LEN,
                       LIM_RETC: INTEGER); EXTERN;
BEGIN
  ( ------------------------------------------------------- )
   (              Query System (Function 2 )                          )
  ( ------------------------------------------------------- )

    FUNC := 2 ;              ( Function number                          )
   LEN    := 35;             ( Length                                   )
    RETC :=     ;            ( Initialize return code variable        )

   HLLPAS  (FUNC,Q_SYS_STR,LEN,RETC);

    Q_SYS_PTR := ADS Q_SYS_STR;        ( Set overlay on string     )

    WITH Q_SYS_PTR DO
     BEGIN
       WRITELN('EHLLAPI version number:              ':23, HLLVERS);
       WRITELN('EHLLAPI level number:                ':23, HLLLVL );
      WRITELN('EHLLAPI  date:                         ':23,  HLLDATE);
      WRITELN('Hardware  base:                       ':23,  HARDW    );
      WRITELN('Program  type:                        ':23,  PTYPE    );
```

```
    WRITELN('Program  version:                    ':23,  PVERS    );
     WRITELN('Component return code:             ':23, COMPRC );
     WRITELN('System return code:                ':23, SYSRC    );
     WRITELN('Model type:                          ':23, MODTYP );
     WRITELN('Monitor type:                        ':23, MONTYP )
   END;

END.
```

C for Windows Guidelines

The following guidelines are helpful in preparing EHLLAPI applications that call EHLLAPI functions in a C language that supports Windows.

Source Code Preparation

Specific statements that you must include in C source code to call EHLLAPI functions are listed in the following text. Also, the syntax for calls to specific EHLLAPI functions is shown.

Required Statements

To call EHLLAPI from a Microsoft C program, the following variables must be declared:

```
HWND hApplWnd;
int   iHLLAPIFunction;
int   iStringLength;
int   iReturnCode;

char zDataString[size];
```

EHLLAPI Function Call Syntax

The syntax for calls to EHLLAPI functions in C is:

```
void FAR PASCAL hllapi( LPINT, LPSTR, LPINT, LPINT );


LockData ( );

hllapi( (LPINT) &iHLLAPIFunction,
          (LPSTR)  zDataString,
          (LPINT)  &iStringLength,
          (LPINT)  &iReturnCode        );

UnlockData ( );
```

or

```
void FAR PASCAL hllapiw( HWND, LPINT, LPSTR, LPINT, LPINT );


LockData ( );

hllapiw (           hApplWnd,
          (LPINT)  &iHLLAPIFunction,
          (LPSTR)  zDataString,
          (LPINT)  &iStringLength,
          (LPINT)  &iReturnCode        );

UnlockData ( );
```

Some PC/3270 and PC400 EHLLAPI functions require preallocation of memory for the return data string parameter. See “Memory Preallocation” on page 17.

Compiling

Compile your PC/3270 or PC400 EHLLAPI application as instructed in *Microsoft Windows Software Development Kit*.

Linking

Link your PC/3270 or PC400 EHLLAPI application as follows:

1. Add either of these instructions to the IMPORT item of the Linker Definition File:

   PCSHLL.hllapi

   PCSHLL.hllapiw

2. Type the following command: LINK d:\path\yourprog/NOD/NOE,,,xLIBCEW+LIBW,d:\path\yourprog.DEF

   This produces an executable module called YOURPROG.EXE. After starting Windows and PC/3270 or PC400, you can execute this C program.

   For more information about compiling and linking programs under Windows, refer to the *Microsoft Windows Software Development Kit*.

Macro Assembler for Windows Guidelines

The following guidelines are helpful in preparing EHLLAPI applications that call EHLLAPI functions in a Macro Assembler language that supports Windows.

Source Code Preparation

Specific statements that you must include in Macro Assembler source code to call EHLLAPI functions are in “Required Statements, General” on page 50. Also, the syntax for calls to EHLLAPI functions is in “EHLLAPI Function Call Syntax” on page 50.

Save any required non–segment registers before calling the HLLAPI subroutine.

After the requested function is executed, control is returned to the next sequential instruction after the HLLAPI subroutine or the HLLAPIW subroutine.

Required Statements, General

Except for the Storage Manager (17) function, calls to EHLLAPI functions from an application written in a Macro Assembler language require a PCB like the one shown below, and statements that set the following registers as shown:

```
HAPPLWND      DW   0          ;Window Handle of
                              ; the application program
FUNC          DW   0          ;function code parameter
                              ; VALUE
DATA_STRING   DB   xxDUP(0)   ;String parameter VALUE
STRLEN        DW   0          ;Length parameter VALUE
RETC          DW   0          ;return code parameter VALUE
```

EHLLAPI Function Call Syntax

The syntax for calls to EHLLAPI functions in Macro Assembler is as follows:

```
PUSH @RETC
PUSH @STRLEN
PUSH @DATA_STRING
PUSH @FUNC
CALL HLLAPI
```

or

```
PUSH @RETC
PUSH @STRLEN
PUSH @DATA_STRING
PUSH @FUNC
PUSH AAPPLWND
CALL HLLAPIW
```

Note: Some PC/3270 and PC400 EHLLAPI functions require preallocation of memory for the return data string parameter. See “Memory Preallocation” on page 17.

Assembling

For Macro Assembler applications that call EHLLAPI functions, assemble the source code as instructed in *Microsoft Windows Software Development Kit*.

Linking

Link your assembled code using the following command:

```
LINK d:\path\yourprog/NOE/NOD,,,xLIBCEW,d:\path\yourdef
```

where d:\path\ is the drive and path specification for the associated file and yourprog is the name of the file containing the compiled object code.

This produces an executable module called YOURPROG.EXE. After starting Windows and PC/3270 or PC400, you can execute this program in the normal manner.

For more details about linking programs under Windows, refer to the *Microsoft Windows Software Development Kit*.

Chapter 6. EHLLAPI Functions

This chapter describes each individual PC/3270 and PC400 EHLLAPI function in detail and explains how to use the EHLLAPI program sampler. The functions are arranged alphabetically by name.

Page Layout Conventions

All EHLLAPI function calls are presented in the same format so that you can quickly retrieve the information you need:

Function Name (Function Number)
- Prerequisite Calls
- Call Parameters
- Return Parameters
- Notes on Using This Function

Prerequisite Calls

This section lists any calls that must be made prior to the function being discussed. If no prerequisite call is required, the word *None* is listed. Prerequisite calls for all EHLLAPI functions are summarized under “Summary of Prerequisite Calls” on page 28.

Call Parameters

This section lists the parameters that must be defined in your program to call the discussed EHLLAPI function and explains how those parameters are to be defined. If a parameter is never used by a function, then *NA* (not applicable) is listed. If a parameter can be overridden by certain values of session parameters defined with calls to the Set Session Parameters (9) function, such session parameters are named.

Return Parameters

These parameters must be received by your program after a call to the discussed EHLLAPI function. This section explains how to interpret those parameters.

Notes on Using This Function

These notes discuss session options that affect the function under discussion. The section also provides technical information about using the function and application development tips.

© Copyright IBM Corp. 1996

Running the EHLLAPI Sampler (PC/3270 DOS Mode Only)

To become familiar with EHLLAPI functions, you can use the EHLLAPI function sampler. The EHLLAPI diskette contains a Program Sampler (PCSSAMP.EXE) that lets you experiment with the various functions.

The Program Sampler allows you to observe the results of calling a specific EHLLAPI function with varying parameters.

Loading and Running the Program Sampler

To run the Program Sampler PCSSAMP.EXE:

1. Have the PC/3270 emulator running.
2. Place the PC/3270 diskette that contains PCSSAMP.EXE in Drive A.
3. Type pcssamp and press Enter.

   This loads PCSSAMP. The EHLLAPI Program Sampler function menu is displayed.
4. Select the desired Program Sampler function menu format by typing either:

   100 for numeric listing of functions
   200 for alphabetic listing of functions
5. Sample the EHLLAPI functions.

   Completing a function returns you to the Program Sampler function menu.

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

   When you are ready to leave the Program Sampler and return to DOS, type 3 , and press Enter from the Program Sampler function menu.

Summary of EHLLAPI Functions

The following table lists the EHLLAPI functions.

Table 4 (Page 1 of 2). EHLLAPI Functions

Number	Function	PC/3270		PC400	Page
		DOS	Windows		
1	Connect Presentation Space (1)	X	X	X	63
2	Disconnect Presentation Space (2)	X	X	X	94
3	Send Key (3)	X	X	X	143
4	Wait (4)	X	X	X	173
5	Copy Presentation Space (5)	X	X	X	81
6	Search Presentation Space (6)	X	X	X	138
7	Query Cursor Location (7)	X	X	X	114
8	Copy Presentation Space to String (8)	X	X	X	85
9	Set Session Parameters (9)	X	X	X	151
10	Query Sessions (10)	X	X	X	120
11	Reserve (11)	X	X	X	134
12	Release (12)	X	X	X	133
13	Copy OIA (13)	X	X	X	73
14	Query Field Attribute (14)	X	X	X	115
15	Copy String to Presentation Space (15)	X	X	X	91
17	Storage Manager (17)	X	–	–	168
18	Pause (18)	X	X	X	108
20	Query System (20)	X	X	X	122
21	Reset System (21)	X	X	X	135
22	Query Session Status (22)	X	X	X	118
23	Start Host Notification (23)	X	X	X	159
24	Query Host Update (24)	X	X	X	117
25	Stop Host Notification (25)	X	X	X	166
30	Search Field (30)	X	X	X	136
31	Find Field Position (31)	X	X	X	98
32	Find Field Length (32)	X	X	X	96
33	Copy String to Field (33)	X	X	X	89
34	Copy Field to String (34)	X	X	X	69
40	Set Cursor (40)	X	X	X	150
41	Start Close Intercept (41)	–	X	X	157
42	Query Close Intercept (42)	–	X	X	111
43	Stop Close Intercept (43)	–	X	X	165
50	Start Keystroke Intercept (50)	X	X	X	161
51	Get Key (51)	X	X	X	101
52	Post Intercept Status (52)	X	X	X	110
53	Stop Keystroke Intercept (53)	X	X	X	167
90	Send File (90)	X	X	X	140
91	Receive File (91)	X	X	X	130
99	Convert Position or Convert RowCol (99)	X	X	X	66

Table 4 (Page 2 of 2). EHLLAPI Functions

Number	Function	PC/3270		PC400	Page
		DOS	Windows		
101	Connect Window Services (101)	–	X	X	65
102	Disconnect Window Service (102)	–	X	X	95
103	Query Window Coordinates (103)	–	X	X	124
104	Window Status (104)	–	X	X	174
105	Change Switch List LT Name (105)	–	X	X	59
106	Change PS Window Name (106)	–	X	X	57
110	Start Playing Macro (110)	–	X	X	164
120	Connect for Structured Fields (120)	X	X	–	61
121	Disconnect from Structured Fields (121)	X	X	–	93
122	Query Communications Buffer Size (122)	X	X	–	112
123	Allocate Communications Buffer (123)	X	X	–	55
124	Free Communications Buffer (124)	X	X	–	100
125	Get Request Completion (125)	X	X	–	105
126	Read Structured Fields (126)	X	X	–	126
127	Write Structured Fields (127)	X	X	–	177

Allocate Communications Buffer (123)

PC/3270		*PC400*
DOS	Windows	
X	X	–

This function obtains a buffer from DOS/Windows management memory. A buffer address must be passed on both the Read SF and Write SF. The EHLLAPI will obtain a buffer from DOS/Windows management memory and place the buffer address into the return parameter string. The requested buffer size (length) is also passed in the parameter string. The buffer size can be from 1 byte to 35 KB (64 KB for the PC/3270 Windows mode) minus 8 bytes in length.

See “Query Communications Buffer Size (122)” on page 112 for information regarding buffer size.

Prerequisite Calls

None.

Call Parameters

Function number	Must be 123.
Data string	See table below.
Data string length	Must be 6.
PS position	NA.

The calling data string can contain:

Byte	Definition
Positions 1–2	16-bit buffer length. (0 < size <= (35 KB – 8) for DOS; 0 < size <= (64 KB – 8) for Windows)
Positions 3–6	32-bit allocated buffer address (returned).

Return Parameters

The following return codes are valid:

Return Code	Explanation
0	The Allocate Communications Buffer function was successful.
2	An error was made in specifying parameters.
9	A system error occurred.
11	Resource unavailable (memory unavailable).

Notes on Using This Function

1. Buffers obtained using this function must not be shared among different processes. If this is attempted, the applications will experience unpredictable results.
2. An EHLLAPI application must issue a Free Communications Buffer (124) function to free the allocated memory.
3. A maximum of 10 buffers can be allocated to an application. If this limit is reached, a return code for resource unavailable (RC=11) will be returned.
4. The Reset System (21) function will free buffers allocated by this function.

Change PS Window Name (106)

PC/3270		*PC400*
DOS	Windows	
–	X	X

This function allows the application to specify a new name for the presentation space window or reset the presentation space window to the default name.

Prerequisite Calls

Connect Window Services (101).

Call Parameters

Function number	Must be 106.
Data string	See table below.
Data string length	Must be specified (normally 3–63 for PC/3270, 4–63 for PC400).
PS position	NA.

The calling data string can contain:

Byte	Definition
Position 1	A 1-character presentation space short session ID.
Position 2	A change request option value, select one of these: X'01' for changing the presentation space window name. X'02' for resetting the presentation space window name.
Positions 3–63	An ASCII string of 1 (for PC/3270) or 2 (for PC400) to 61 bytes including a terminator byte. The ASCII string must end with a NULL character. This string must contain at least one non-NULL character followed by a NULL character.

Return Parameters

The following return codes are valid:

Return Code	Explanation
0	The Change PS Window Name function was successful.
1	An incorrect host presentation space short session ID was specified or was not connected.
2	An error was made in specifying parameters.
9	A system error occurred.
12	The session stopped.

Notes on Using This Function

1. A string is ended at the first NULL character found.
2. The NULL character overrides the specified string length.
3. If the NULL character is not at the end of the specified length, the last byte at the specified length is replaced by a NULL character, and the remainder of the data string is lost.
4. If the NULL character is found before the specified length, the string is truncated at that point, and the remainder of the data string is lost.
5. If the application fails to reset the presentation space name before exiting, the exit list processing resets the name.

Change Switch List LT Name (105)

PC/3270		*PC400*
DOS	Windows	
–	X	X

This function allows the application to change or reset a switch list for a selected logical terminal (LT). The application must specify on the call the name to be inserted in the switch list.

Note: This is for compatibility with OS/2 EHLLAPI. The result is the same as the Change PS Window Name (106) function.

Prerequisite Calls

Connect Window Services (101).

Call Parameters

Function number	Must be 105.
Data string	See table below.
Data string length	Must be specified (normally 4–63).
PS position	NA.

The calling data string can contain:

Byte	Definition
Position 1	A 1-character presentation space short session ID.
Position 2	A change request option value, select one of these: X'01' for changing switch list LT name. X'02' for resetting switch list LT name.
Positions 3–63	An ASCII string of 2 to 61 bytes including a terminator byte. The ASCII string must end with a NULL character. This string must contain at least one non-NULL character followed by a NULL character.

Return Parameters

The following return codes are valid:

Return Code	Explanation
0	The Change Switch List LT Name function was successful.
1	An incorrect host presentation space short session ID was specified or was connected.
2	An error was made in specifying parameters.
9	A system error occurred.
12	The session stopped.

Notes on Using This Function

1. A string is ended at the first NULL character found.
2. The NULL character overrides the specified string length.
3. If the NULL character is not at the end of the specified length, the last byte at the specified length is replaced by a NULL character, and the remainder of the data string is lost.
4. If the NULL character is found before the specified length, the string is truncated at that point, and the remainder of the data string is lost.
5. If the application fails to reset the switch list LT name before exiting, the exit list processing resets the name.

Connect for Structured Fields (120)

PC/3270		PC400
DOS	Windows	
X	X	–

This function allows an application to establish a connection to the Emulation Program to send and receive Structured Field data to/from a host application. The PC application must provide the Query Reply data field and must point to it with in the parameter string. The destination/origin ID returned by the emulator will be returned to the application.

Prerequisite Calls

None.

Call Parameters

Function number	Must be 120.
Data string	See table below.
Data string length	Length of string (7 or 11).
PS position	NA.

The calling data string can contain:

Byte	Definition
Position 1	A 1-character presentation space short session ID.
Positions 2–5	Address of the Query Reply data buffer.
Positions 6–7	Destination/Origin Unique ID. (16-bit word, returned)
Positions 8–11	The data in these position is ignored by EHLLAPI. However, no error is caused if the migrating program has data in these positions. This data is accepted to provide compatibility with migrating applications.

Return Parameters

The following return codes are valid:

Return Code	Explanation
0	The Connect for Structured Fields function was successful.
1	A specified host presentation space short session ID was incorrect or was not connected.
2	An error was made in specifying parameters.
9	A system error occurred.
10	The function was not supported by the emulation program.
32	An application has already connected to this session for communications (successful connect).
39	One DDM session is already connected to this session.

Notes on Using This Function

1. EHLLAPI scans the query reply buffers for the destination/origin ID (DOID) self-defining parameter (SDP) to determine the contents of the DOID field of the query reply. If this value is X'0000', the emulator will assign a DOID to the application and EHLLAPI will fill in the DOID field of the query reply with the assigned ID. If the value specified by the application in the DOID field of the query reply is a nonzero value, the emulator will assign the specified value as the application's DOID, assuming that the ID has not been previously assigned. In the event that the specified DOID is already in use, a return code of 2 is returned by EHLLAPI.
2. The application should build the Query Reply data structures in the application's private memory. Refer to Appendix F, "Query Reply Data Structures Supported by EHLLAPI" for the detailed formats and usages of the query reply data structures supported by EHLLAPI.
3. Only cursory checking is performed on the Query Reply data. Only the ID and the length of the structure are checked for validity.
4. Only one DDM base type connect is allowed per host session. If the DDM connection supports the self-defining parameter (SDP) for the destination origin ID (DOID), then multiple connects are allowed.
5. If return code RC=32 or RC=39 is received, an application is already connected to the selected session and use of that presentation space should be approached with caution. Conflicts with SRPI, File Transfer, and other EHLLAPI applications might result.

Connect Presentation Space (1)

PC/3270		PC400
DOS	Windows	
X	X	X

This function establishes a connection between your EHLLAPI application program and the host presentation space.

Prerequisite Calls

None.

Call Parameters

Function number	Must be 1.
Data string	One-character short name of the host presentation space.
Length	NA (1 is implied).
PS position	NA.

Return Parameters

The following return codes are valid:

Return Code	Explanation
0	The Connect Presentation Space function was successful; the host presentation space is unlocked and ready for input.
1	An incorrect host presentation space ID was specified. Specified session either does not exist or is a logical printer session.
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.

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.

Notes on Using This Function

1. The Connect Presentation Space function is affected by the CONLOG/CONPHYS session option. Refer to item 11 on page 154 for more information.
2. An EHLLAPI application cannot be connected to multiple presentation spaces concurrently; however, the connections are sequential and the application cannot go back and forth between connections. Calls requiring the Connect Presentation Space function as a prerequisite use the currently connected presentation space. For example, if an application is connected to presentation space A, B, and C in that order, the application must connect to B or A again in order to issue functions.
3. If an EHLLAPI program connects to the presentation space, other EHLLAPI programs are prevented from using this presentation space. After one of the following actions, another EHLLAPI programs can access this presentation space.

   Exit the EHLLAPI program that is currently connected.

   Connect the current EHLLAPI program to the other presentation spaces. If the EHLLAPI program is connected to multiple presentation spaces, these presentation spaces without the latest one can be accessed by another EHLLAPI program.

   Use the Disconnect Presentation Space (2) function to drop the connection.
4. The Reset System (21) function reinitializes EHLLAPI program to its starting state.
5. Table 2 on page 28 shows the functions that need a Connect Presentation Space call as a prerequisite call.
6. To exist, a session must be declared during customization. You cannot connect to a session that is not declared or to a session that is defined as a logical printer session. See the *Personal Communications Version 4.1 Up and Running for DOS (Full-Function Mode)*, the *Personal Communications Version 4.1 Up and Running for Windows*, the *Personal Communications Version 4.1 Reference for Windows*, or the *Personal Communications AS/400 and 3270 Version 4.1 Up and Running for Windows*.

Connect Window Services (101)

PC/3270		PC400
DOS	Windows	
–	X	X

This function allows the application to manage the presentation space windows. Only one EHLLAPI application at a time can be connected to a presentation space for window services.

An EHLLAPI application can connect to more than one presentation space concurrently for window services.

Prerequisite Calls

None.

Call Parameters

Function number	Must be 101.
Data string	One-character short session ID of the host presentation space.
Data string length	NA (1 is implied).
PS position	NA.

Return Parameters

The following return codes are valid:

Return Code	Explanation
0	The Connect Window Services function was successful.
1	An incorrect host presentation space short session ID was specified or was not connected.
9	A system error occurred.
10	The function was not supported by the emulation program.
11	This resource is unavailable. The host presentation space is being used by another system function.

Notes on Using This Function

1. An EHLLAPI application can be connected to multiple presentation space windows at the same time. The application can go back and forth between the connected presentation space windows without having to disconnect. For example, if an application is connected to presentation space windows A, B, and C, the application can access all of A, B, and C at the same time, and the other applications cannot access A, B, or C.
2. If an EHLLAPI program connects to the presentation space window, other EHLLAPI programs are prevented from using this presentation space window. After one of the following actions, another EHLLAPI program can access this presentation space window.

   Exit the EHLLAPI program that is currently connected.

   Use the Disconnect Window Service (102) function to drop the connection.
3. The Reset System (21) function reinitializes the EHLLAPI program to its starting state.

Convert Position or Convert RowCol (99)

PC/3270		*PC400*
DOS	Windows	
X	X	X

These functions convert the host presentation space positional value into the display row-column coordinates or convert the display row-column coordinates into the host presentation space positional value. These functions do not change the cursor position.

Prerequisite Calls

None.

Call Parameters

Function number
: Must be 99.

Data string
: Host presentation space short name and P for the Convert Position function. For example, AP converts the presentation space position of session A.

OR

Host presentation space short name and R for the Convert RowCol function. For example, AR converts the row and column coordinates of session A.

Length
: Row when R, above, is specified. Lower limit for valid input is 1. The upper limit for valid input depends on how your host presentation space is configured. See “Notes on Using This Function” on page 68.

NA when P is specified for data string.

PS position
: Column when R, above, is specified. Lower limit for valid input is 1. The upper limit for valid input range is 24–43 depending on how your host presentation space is configured. See “Notes on Using This Function” on page 68.

Host presentation space position when P, above, is specified. Lower limit for valid input is 1. The upper limit for valid input range is 1920–3564 depending on how your host presentation space is configured. See “Notes on Using This Function” on page 68.

Return Parameters

This function returns a length and a return code.

Length — For the Convert Position function (P as second character in calling data string), a number between 1 and 43 (for PC/3270) or 27 (for PC400) is returned. This value is the number of the row that contains the PS position contained in the calling PS position parameter. The upper limit can be smaller than 43 (for PC/3270) or 27 (for PC400) depending on how the host presentation space is configured.

For the Convert RowCol function (R as second character in calling data string), a value of 0 indicates an error in the input value for row (calling length parameter).

Return code — The Convert Position or RowCol function is the exception to the rule that the fourth return parameter always contains a return code. For this function, the value returned in the fourth parameter is called a status code. This status code can contain data or a return code. Your application must provide for processing of this status code to prevent unpredictable results or an error.

If the value of the fourth parameter is 0, 9998, or 9999, it is a return code.

For the Convert Position function (P as second character of calling data string), a value in the range of 1–132 is the number of the column that contains the PS position passed in the calling PS Position parameter. The upper limit can be smaller than 132 depending on how the host presentation space is configured.

For the Convert RowCol function (R as second character of calling data string), a value in the range of 1–3564 represents the host presentation space position that corresponds to the row and column values passed in the calling length and PS position parameters, respectively. The upper limit can be smaller than 3564 depending on how the host presentation space is configured.

The following status codes are valid:

Status Code	Explanation
0	This is an incorrect PS position or column.
>0	This is the PS position or column.
9998	An incorrect host presentation space ID was specified or a system error occurred.
9999	Character 2 in the data string is not P or R.

Notes on Using This Function

1. To configure your presentation space, refer to the *Personal Communications Version 4.1 Up and Running for DOS (Full-Function Mode)*, the *Personal Communications Version 4.1 Up and Running for Windows*, the *Personal Communications Version 4.1 Reference for Windows*, or the *Personal Communications AS/400 and 3270 Version 4.1 Up and Running for Windows*.
2. To find out how many rows and columns are in your presentation space, examine bytes 12 and 14 of the returned data string parameter for the Query Session Status (22) function. See “Query Session Status (22)” on page 118.

Copy Field to String (34)

PC/3270		PC400
DOS	Windows	
X	X	X

This function transfers characters from a field in the host-connected presentation space into a string.

The Copy Field to String function translates the characters in the host source presentation space into American National Standard Code for Information Interchange (ASCII). Attribute bytes and other characters not represented in ASCII normally are translated into blanks.

Prerequisite Calls

Connect Presentation Space (1).

Call Parameters

Function number — Must be 34.

Data string — Preallocated target data string. When the Set Session Parameters (9) function with extended attribute bytes (EAB) option is issued, the length of the data string must be at least twice the length of the field.

Length — Number of bytes to copy (length of the data string).

PS position — Identifies the target field. Might be the PS position of any byte within the target field. Copy always starts at the beginning of the field.

Return Parameters

This function returns a data string and a return code.

Data string — A string containing data from the identified field in the host PS. The first byte in the returned data string is the beginning byte of the identified field in the host PS. The number of bytes in the returned data string is determined by the smaller of:

- Number of bytes specified in the calling length parameter
- Number of bytes in the identified field in the host PS

Return code — The following codes are valid:

Return Code	Explanation
0	The Copy Field to String function was successful.
1	Your program is not connected to a host session.
2	An error was made in specifying parameters.
6	The data to be copied and the target field are not the same size. The data is truncated if the string length is smaller than the field copied.
7	The host presentation space position is incorrect.
9	A system error was encountered.
24	Unformatted host PS.

Notes on Using This Function

1. The field position and length information can be found by using the Find Field Position (31) and Find Field Length (32) functions. The Copy Field to String function can be used with either protected or unprotected fields, but only in a *field-formatted* host presentation space.

2. The copy is ended either when:

   The end of the field is reached.
   The length of the target string is exceeded.

   When the field wraps at the end of the presentation space, wrapping occurs when the end of the PS is reached.

3. An EAB can be returned when the Set Session Parameters (9) function EAB option is used. The EAB is related to each character in the presentation space and is returned preceding each character.

   The Copy Field to String function is affected by the EAB/NOEAB, the XLATE/NOXLATE, and the ATTRB/NOATTRB session options. Refer to item 5 on page 153 and items 14 and15 on page 155 for more information.

   If the target data string provided is not large enough to hold the requested number of bytes, the copy ends successfully when the end of the target data string is reached.

   As previously stated, the return of attributes by the various Copy (5, 8, and 34) functions is affected by the Set Session Parameters (9) function. The involved set session parameters have the following effect:

   Set Session Parameter Effect on the COPY Function

NOEAB	Attributes are not returned. Only text is copied from the presentation space to the user buffer.
EAB and NOXLATE	Attributes are returned as defined in the following tables.
EAB and XLATE	The colors used for the presentation space display are returned. Colors can be remapped; so the attribute colors are not the ones returned by the **COPY** functions when XLATE and EAB are on at the same time.

   The returned character attributes are defined in the following tables. The attribute bit positions are in IBM format with bit 0 the leftmost bit in the byte.

   3270 character attributes are returned from the host to the emulator. The data in the following table applies when EAB and NOXLATE are set.

Bit Position	Meaning
0–1	Character highlighting 00 = Normal 01 = Blink (always 0 since Presentation Manager does not support blink) 10 = Reverse video 11 = Underline

Bit Position	Meaning
2–4	Character color (Color remap can override this color definition.) 000 = Default 001 = Blue 010 = Red 011 = Pink 100 = Green 101 = Turquoise 110 = Yellow 111 = White
5–7	Reserved

5250 character attributes are returned from the host to the emulator. The data in the following table applies when EAB and NOXLATE are set.

Bit Position	Meaning
0	Reverse image 0 = Normal image 1 = Reverse image
1	Underline 0 = No underline 1 = Underline
2	Blink 0 = Not blink 1 = Blink
3	Separator of columns 0 = No separator 1 = Separator
4–7	Reserved

The following table shows PC/3270 and PC400 character color attributes. and applies when EAB and XLATE are set.

Bit Position	Meaning
0–3	Background character colors 0000 = Black 0001 = Blue 0010 = Green 0011 = Cyan 0100 = Red 0101 = Magenta 0110 = Brown (3270), Yellow (5250) 0111 = White

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

For PS/2 monochrome displays, the characters in the application (PC) session appear as various shades of gray. This is required to give users their remapped colors in the EHLLAPI application session so they can get what they see in their host application presentation spaces.

4. To use this function, you must preallocate memory to receive the returned data string parameter. The statements required to preallocate this memory vary depending on the language in which your application is written. Refer to “Memory Preallocation” on page 17 for more information.

Copy OIA (13)

PC/3270		PC400
DOS	Windows	
X	X	X

This function returns the current operator information area (OIA) data from the host-connected presentation space.

The OIA is located under the bottom dividing line of the screen and is used to display session status information about the connection between the personal computer and the host.

Prerequisite Calls

Connect Presentation Space (1).

Call Parameters

Function number	Must be 13.
Data string	Preallocated target data string.
Length	103 bytes.
PS position	NA.

Return Parameters

This function returns a data string and a return code.

Data string	A 103-byte string. See “Format of the Returned OIA Data String” on page 74 for more information.
Return code	The following codes are valid:

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

Notes on Using This Function

1. The OIA Group consists of the bits that show the status of the connected sessions. The group is categorized by the represented host function. (For example, Group 8 consists of the bits that show all conditions of the input inhibit in the session.) The states of each group are ordered so that the high-order bits represent the indicators of higher priority. That is, bit 7 has priority over bit 0. Therefore, if more than one state is active within a group, the state with the highest priority is the active state within that group.
2. To use this function, preallocate memory to receive the returned data string parameter. The statements required to preallocate this memory vary depending on the language in which your application is written. Refer to “Memory Preallocation” on page 17 for more information.

Format of the Returned OIA Data String

The OIA data string contains the following information.

Byte	Definition
Position 1	The OIA Format Byte. The value is 9.
Position 2–81	The OIA Image in the host code points.
Position 82–103	OIA Group Indicator Meanings.

PC/3270 OIA Group Indicator Meanings and its Image: The OIA Image Group consists of an 80-byte ASCII character string with no attribute bytes that contains the OIA image in host code points. Figure 2 shows the hexadecimal codes found in the host presentation space and the characters they represent. The returned data can be translated into OIA graphics characters. Refer to the *Personal Communications Version 4.1 Up and Running for DOS (Full-Function Mode)*, *Personal Communications Version 4.1 Up and Running for Windows*, or the *Personal Communications AS/400 and 3270 Version 4.1 Up and Running for Windows* for information on the OIA indicators.

To translate the returned data into OIA graphics characters, proceed as follows:

1. Print the data returned in bytes 2–81 to the screen or to a printer.
2. Using the code page chart applicable to the device on which the output appears, find the hexadecimal value corresponding to each character.
3. Using Figure 2, find the OIA graphics character corresponding to each hexadecimal value found in step 2.

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

Figure 2. Host Presentation Space Characters

Group 1 (Offset 82): Online and screen ownership

Bit	Meaning
0–1	Reserved
2	SSCP-LU session owns screen
3	LU-LU session owns screen
4	Online and not owned
5	Subsystem ready
6–7	Reserved

Group 2 (Offset 83): Character selection

Bit	Meaning
0	Reserved
1	APL
2–7	Reserved

Group 3 (Offset 84): Shift state

Bit	Meaning
0	Upper shift
1	Numeric
2	CAPS
3–7	Reserved

Group 4 (Offset 85): PSS group 1

Bit	Meaning
0–7	Reserved

Group 5 (Offset 86): Highlight group 1

Bit	Meaning
0	Operator selectable
1	Field inherit
2–7	Reserved

Group 6 (Offset 87): Color group 1

Bit	Meaning
0	Operator selectable
1	Field inherit
2–7	Reserved

Group 7 (Offset 88): Insert

Bit	Meaning
0	Insert mode
1–7	Reserved

Group 8 (Offset 89–93): Input inhibited (5 bytes)

– Byte 1 (Offset 89)

Bit	Meaning
0	Non-resettable machine check
1	Reserved
2	Machine check
3	Communications check
4	Program check
5–7	Reserved

– Byte 2 (Offset 90)

Bit	Meaning
0	Device busy
1–2	Reserved
3	Minus function
4	Too much entered
5–7	Reserved

– Byte 3 (Offset 91)

Bit	Meaning
0–2	Reserved
3	Incorrect dead key combination. Limited key.
4	Wrong place
5–7	Reserved

– Byte 4 (Offset 92)

Bit	Meaning
0–1	Reserved
2	System wait
3–7	Reserved

– Byte 5 (Offset 93)

Bit	Meaning
0–7	Reserved

Group 9 (Offset 94): PSS group 2

Bit	Meaning
0–7	Reserved

Group 10 (Offset 95): Highlight group 2

Bit	Meaning
0–7	Reserved

Group 11 (Offset 96): Color group 2

Bit	Meaning
0–7	Reserved

Group 12 (Offset 97): Communication error reminder

Bit	Meaning
0	Communications error
1–7	Reserved

Group 13 (Offset 98): Printer state

Bit	Meaning
0–7	Reserved

Group 14 (Offset 99): Graphics

Bit	Meaning
0–7	Reserved

Group 15 (Offset 100): Reserved

Group 16 (Offset 101): Automatic key play/record state

Bit	Meaning
0–7	Reserved

Group 17 (Offset 102): Automatic key quit/stop state

Bit	Meaning
0–7	Reserved

Group 18 (Offset 103): Expanded state

Bit	Meaning
0–7	Reserved

PC400 OIA Group Indicator Meanings and its Image: Details of the OIA Group are listed in the following tables.

Group 1 (Offset 82): Online and screen ownership

Bit	Meaning	Beginning Position of Data String
0–2	Reserved	
3	System available	1
4	Reserved	
5	Subsystem ready	
6–7	Reserved	

Group 2 (Offset 83): Character selection

Bit	Meaning	Beginning Position of Data String
0–1	Reserved	
2	Katakana (Japan only)	
3	Alphanumeric	
4–5	Reserved	
6	Hiragana (Japan only)	
7	Double-byte character	

Group 3 (Offset 84): Shift state

Bit	Meaning	Beginning Position of Data String
0	Reserved	
1	Keyboard shift	39
2	CAPS	
3–6	Reserved	
7	Double-byte character input available	

Group 4 (Offset 85): PSS group 1

Bit	Meaning	Beginning Position of Data String
0–7	Reserved	

Group 5 (Offset 86): Highlight group 1

Bit	Meaning	Beginning Position of Data String
0–7	Reserved	

Group 6 (Offset 87): Color group 1

Bit	Meaning	Beginning Position of Data String
0–7	Reserved	

Group 7 (Offset 88): Insert

Bit	Meaning	Beginning Position of Data String
0	Insert mode	68
1–7	Reserved	

Group 8 (Offset 89–93): Input inhibited (5 bytes)

– Byte 1 (Offset 89)

Bit	Meaning	Beginning Position of Data String
0–7	Reserved	

– Byte 2 (Offset 90)

Bit	Meaning	Beginning Position of Data String
0–7	Reserved	

– Byte 3 (Offset 91)

Bit	Meaning	Beginning Position of Data String
0–4	Reserved	
5	Operator input error	64
6–7	Reserved	

– Byte 4 (Offset 92)

Bit	Meaning	Beginning Position of Data String
0–1	Reserved	
2	System wait	64
3–7	Reserved	

– Byte 5 (Offset 93)

Bit	Meaning	Beginning Position of Data String
0–7	Reserved	

Group 9 (Offset 94): PSS group 2

Bit	Meaning	Beginning Position of Data String
0–7	Reserved	

Group 10 (Offset 95): Highlight group 2

Bit	Meaning	Beginning Position of Data String
0–7	Reserved	

Group 11 (Offset 96): Color group 2

Bit	Meaning	Beginning Position of Data String
0–7	Reserved	

Group 12 (Offset 97): Communication error reminder

Bit	Meaning	Beginning Position of Data String
0–6	Reserved	
7	Message wait	3

Group 13 (Offset 98): Printer state

Bit	Meaning	Beginning Position of Data String
0–7	Reserved	

Group 14 (Offset 99): Graphics

Bit	Meaning	Beginning Position of Data String
0–7	Reserved	

Group 15 (Offset 100): Reserved

Group 16 (Offset 101): Automatic key play/record state

Bit	Meaning	Beginning Position of Data String
0–7	Reserved	

Group 17 (Offset 102): Automatic key quit/stop state

Bit	Meaning	Beginning Position of Data String
0–7	Reserved	

Group 18 (Offset 103): Expanded state

Bit	Meaning	Beginning Position of Data String
0–7	Reserved	

Copy Presentation Space (5)

PC/3270		*PC400*
DOS	Windows	
X	X	X

This function copies the contents of the host-connected presentation space into a data string that you define in your EHLLAPI application program.

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

Prerequisite Calls

Connect Presentation Space (1).

Call Parameters

Function number	Must be 5.
Data string	Preallocated target string the size of your host presentation space. The value can vary depending on how your host presentation space is configured. When the Set Session Parameters (9) function with the EAB option is issued, the length of the data string must be at least twice the field length.
Length	NA (length of the host PS is implied).
PS position	NA.

Return Parameters

This function returns a data string and a return code.

Data string	Contents of the connected host presentation space.
Return code	The following codes are valid:

Return Code	Explanation
0	The host presentation space contents were copied to the application program. The target presentation space was active, and the keyboard was unlocked.
1	Your program is not connected to a host session.
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

1. An EAB can be returned when the Set Session Parameters (9) function EAB option is used. EAB is related to each character in the presentation space and is returned preceding each character.
2. The Copy Presentation Space function is affected by the ATTRB/NOATTRB, the EAB/NOEAB, and the XLATE/NOXLATE session options. Refer to item 5 on page 153 and items 14 and 15 on page 155 for more information.

   If the target data string provided is not large enough to hold the requested number of bytes, the copy ends successfully when the end of the target data string is reached.

   As previously stated, the return of attributes by the various Copy (5, 8, and 34) functions is affected by the Set Session Parameters (9) function. The involved set session parameters have the following effect:

   Set Session Parameter Effect on the Copy Function

NOEAB	Attributes are not returned. Only text is copied from the presentation space to the user buffer.
EAB and NOXLATE	Attributes are returned as defined in the following tables.
EAB and XLATE	The colors used for the presentation space display are returned. Colors can be remapped; so the attribute colors are not the ones returned by the Copy functions when XLATE and EAB are on at the same time.

   The returned character attributes are defined in the following tables. The attribute bit positions are in IBM format with bit 0 the leftmost bit in the byte.

   3270 character attributes are returned from the host to the emulator. The following table applies when EAB and NOXLATE are set.

Bit Position	Meaning
0–1	Character highlighting 00 = Normal 01 = Blink (always 0 since Presentation Manager does not support blink) 10 = Reverse video 11 = Underline

Bit Position	Meaning
2–4	Character color (Color remap can override this color definition.) 000 = Default 001 = Blue 010 = Red 011 = Pink 100 = Green 101 = Turquoise 110 = Yellow 111 = White
5–7	Reserved

5250 character attributes are returned from the host to the emulator. The following table applies when EAB and NOXLATE are set.

Bit Position	Meaning
0	Reverse image 0 = Normal image 1 = Reverse image
1	Underline 0 = No underline 1 = Underline
2	Blink 0 = Not blink 1 = Blink
3	Separator of columns 0 = No separator 1 = Separator
4–7	Reserved

The following table shows PC/3270 and PC400 character color attributes and applies when EAB and XLATE are set.

Bit Position	Meaning
0–3	Background character colors 0000 = Black 0001 = Blue 0010 = Green 0011 = Cyan 0100 = Red 0101 = Magenta 0110 = Brown (3270), Yellow (5250) 0111 = White

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

For PS/2 monochrome displays, the characters in the application (PC) session appear as various shades of gray. This is required to give users their remapped colors in the EHLLAPI application session so they can get what they see in their host application presentation spaces.

3. If you want to copy only a portion of the host presentation space, use the Copy Presentation Space to String (8) function.
4. To use this function, preallocate memory to receive the returned data string parameter. The statements required to preallocate this memory vary depending on the language in which your application is written. Refer to “Memory Preallocation” on page 17 for more information.

Copy Presentation Space to String (8)

PC/3270		*PC400*
DOS	Windows	
X	X	X

This function is used to copy all or part of the host-connected presentation space into a data string that you define in your EHLLAPI application program.

The input PS position is the offset into the host presentation space. This offset is based on a layout in which the upper-left corner (row 1, column 1) is Location 1 and the bottom right corner is 3564, which is the maximum screen size for the host presentation space. The value of PS Position + (Length – 1) cannot exceed the configured size of your host presentation space.

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

Prerequisite Calls

Connect Presentation Space (1).

Call Parameters

Function number	Must be 8.
Data string	Preallocated target data string. When the Set Session Parameters (9) function with the EAB option is issued, the length of the data string must be at least twice the length of the field.
Length	Length of the target data string.
PS position	Position within the host presentation space of the first byte in your target data string.

Return Parameters

This function returns a data string and a return code.

Data string	Contents of the host PS.
Return code	The following codes are valid:

Return Code	Explanation
0	The host presentation space contents were copied to the application program. The target presentation space was active, and the keyboard was unlocked.
1	Your program is not connected to a host session.
2	An error was made in specifying string length, or the sum of (Length – 1) + PS position is greater than the size of the connected host PS.
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 incorrect.
9	A system error was encountered.

Notes on Using This Function

1. An EAB can be returned when the Set Session Parameters (9) function EAB option is used. EAB is related to each character in the presentation space and is returned preceding each character.
2. The Copy Presentation Space to String function is affected by the ATTRB/NOATTRB, the EAB/NOEAB, and the XLATE/NOXLATE session options. Refer to item 5 on page 153 and items 14 and 15 on page 155 for more information.

   If the target data string provided is not large enough to hold the requested number of bytes, the copy ends successfully (RC=0, 4, or 5) when the end of the target data string is reached.

   As previously stated, the return of attributes by the various Copy (5, 8, and 34) functions is affected by the Set Session Parameters (9) function. The involved set session parameters have the following effect:

   Set Session Parameter Effect on the Copy Function

NOEAB	Attributes are not returned. Only text is copied from the presentation space to the user buffer.
EAB and NOXLATE	Attributes are returned as defined in the following tables.
EAB and XLATE	The colors used for the presentation space display are returned. Colors can be remapped; so the attribute colors are not the ones returned by the Copy functions when XLATE and EAB are on at the same time.

   The returned character attributes are defined in the following tables. The attribute bit positions are in IBM format with bit 0 the leftmost bit in the byte.

   3270 character attributes are returned from the host to the emulator. The following table applies when EAB and NOXLATE are set.

Bit Position	Meaning
0–1	Character highlighting 00 = Normal 01 = Blink (always 0 since Presentation Manager does not support blink) 10 = Reverse video 11 = Underline
2–4	Character color (Color remap can override this color definition.) 000 = Default 001 = Blue 010 = Red 011 = Pink 100 = Green 101 = Turquoise 110 = Yellow 111 = White
5–7	Reserved

5250 character attributes are returned from the host to the emulator. The following table applies when EAB and NOXLATE are set.

Bit Position	Meaning
0	Reverse image 0 = Normal image 1 = Reverse image
1	Underline 0 = No underline 1 = Underline
2	Blink 0 = Not blink 1 = Blink
3	Separator of columns 0 = No separator 1 = Separator
4–7	Reserved

This table shows PC/3270 and PC400 character color attributes and applies when EAB and XLATE are set.

Bit Position	Meaning
0–3	Background character colors 0000 = Black 0001 = Blue 0010 = Green 0011 = Cyan 0100 = Red 0101 = Magenta 0110 = Brown (3270), Yellow (5250) 0111 = White

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

For PS/2 monochrome displays, the characters in the application (PC) session appear as various shades of gray. This is required to give users their remapped colors in the EHLLAPI application session so they can get what they see in their host application presentation spaces.

3. To use this function, preallocate memory to receive the returned data string parameter. The statements required to preallocate this memory vary depending on the language in which your application is written. Refer to “Memory Preallocation” on page 17 for more information.

Copy String to Field (33)

PC/3270		*PC400*
DOS	Windows	
X	X	X

This 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.

Prerequisite Calls

Connect Presentation Space (1).

Call Parameters

Function number	Must be 33.
Data string	String containing the data to be transferred to a target field in the host presentation space.
Length	Length, in number of bytes, of the source data string. Overridden if in EOT mode.
PS Position	Identifies the target field. It might be the PS position of any byte within the target field. Copy always starts at the beginning of the field.

Return Parameters

This function returns these valid return codes:

Return Code	Explanation
0	The Copy String to Field function was successful.
1	Your program is not connected to a host session.
2	Parameter error or zero length for copy.
5	The target field was protected or inhibited, or data that is not valid 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 incorrect.
9	A system error was encountered.
24	Unformatted host PS.

Notes on Using This Function

1. The Copy String to Field function is affected by the STRLEN/STREOT and EOT session options. Refer to items 1 and 2 on page 152 for more information.
2. The string to be transferred is specified with the calling data string parameter. The string ends when one of these three conditions is encountered:

   When an end-of-text (EOT) delimiter is encountered in the string if EOT mode was selected using the Set Session Parameters (9) function. (See "Set Session Parameters (9)" on page 151).

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

   When an end-of-field is encountered in the field.

   Note: If the field at the end of the host presentation space wraps, wrapping occurs when the end of the PS is reached.
3. The keyboard mnemonics (see Send Key (3) function) cannot be sent using the Copy String to Field function.
4. The first byte of the data to be transferred is always placed at the beginning of the field that contains the specified PS position.

Copy String to Presentation Space (15)

PC/3270		*PC400*
DOS	Windows	
X	X	X

This function copies an ASCII data string directly into the host presentation space at the location specified by the PS position calling parameter.

Prerequisite Calls

Connect Presentation Space (1).

Call Parameters

Function number	Must be 15.
Data string	String of ASCII data to be copied into the host presentation space.
Length	Length, in number of bytes, of the source data string. Overridden if in EOT mode.
PS position	Position in the host presentation space to begin the copy, a value between 1 and the configured size of your host presentation space.

Return Parameters

This function returns one of these valid return codes:

Return Code	Explanation
0	The Copy String to Presentation Space function was successful.
1	Your program is not connected to a host session.
2	Parameter error or zero length for copy.
5	The target presentation space is protected or inhibited, or data that is not valid 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 incorrect.
9	A system error was encountered.

Notes on Using This Function

1. The Copy String to Presentation Space function is affected by the STRLEN/STREOT and the EOT session options. Refer to items 1 and 2 on page 152 for more information.
2. The keyboard mnemonics (see Send Key (3) function) cannot be sent using the Copy String to Presentation Space function.
3. The string ends when an end-of-text (EOT) delimiter is encountered in the string if EOT mode was selected using the Set Session Parameters (9) function. See “Set Session Parameters (9)” on page 151.
4. Although the Send Key (3) function accomplishes the same purpose, this function responses with the prompt and enters a command more quickly. Because the Send Key (3) function emulates the terminal operator typing the data from the keyboard, its process speed is slow for an application operating a lot of data. This function provides a faster input path to the host.
5. The original data (the copied string) cannot exceed the size of the presentation space.
6. If there is a field attribute or a protected field in the target area, the string and the area are shifted by an amount that equals their length.

Disconnect from Structured Fields (121)

PC/3270		*PC400*
DOS	Windows	
X	X	–

This function drops the connection between the Emulation Program and the EHLLAPI application. The EHLLAPI application must disconnect from the Emulation Program before exiting from the system. The EHLLAPI application should issue this function request.

The Reset System (21) function will also disconnect any outstanding SF connections.

Prerequisite Calls

Connect for Structured Fields (120).

Call Parameters

Function number	Must be 121.
Data string	See the following table.
Data string length	Must be 3.
PS position	NA.

The calling data string can contain:

Byte	Definition
Position 1	A 1-character presentation space short session ID.
Positions 2–3	Destination/Origin Unique ID returned by the Connect for Structured Fields (120) function.

Return Parameters

The following return codes are valid:

Return Code	Explanation
0	The Disconnect from Structured Fields function was successful.
1	A specified host presentation space short session ID was not valid or was not connected.
2	An error was made in specifying parameters.
9	A system error occurred.
40	Disconnected with asynchronous requests pending.

Notes on Using This Function

1. Any outstanding asynchronous Read SF or Write SF function requests existing when a Disconnect from D/O Structured Fields function is called will be returned if the application issues the Get Request Completion function call (use the asynchronous form of this function when “cleaning up” following a Disconnect).
2. The Reset System (21) function will also free any outstanding asynchronous requests (requests that have not been retrieved by the application using the Get Request Completion (125) function).

Disconnect Presentation Space (2)

PC/3270		*PC400*
DOS	Windows	
X	X	X

This function drops the connection between your EHLLAPI application program and the host presentation space. Also, if a host presentation space is reserved using the Reserve (11) function, it is released upon execution of the Disconnect Presentation Space function.

Prerequisite Calls

Connect Presentation Space (1).

Call Parameters

Function number	Must be 2.
Data string	NA.
Length	NA.
PS position	NA.

Return Parameters

This function returns these valid return codes:

Return Code	Explanation
0	The Disconnect Presentation Space function was successful.
1	Your program is not connected to the host presentation space.
9	A system error was encountered.

Notes on Using This Function

1. After the Disconnect Presentation Space function is called, functions that interact with the host-connected presentation space are no longer valid (for example, Send Key (3), Wait (4), Reserve (11) and Release (12) functions).
2. Your EHLLAPI application should disconnect from the host presentation space before exiting.
3. The Disconnect Presentation Space function does not reset the session parameters to the defaults. Your EHLLAPI application must call the Reset System (21) function to accomplish this.

Disconnect Window Service (102)

PC/3270		*PC400*
DOS	Windows	
–	X	X

This function disconnects the window services connection between the EHLLAPI program and the specified host presentation space window.

Prerequisite Calls

Connect Window Services (101).

Call Parameters

Function number	Must be 102.
Data string	One-character short session ID of the host presentation space.
Data string length	NA (1 is implied).
PS position	NA.

Return Parameters

The following return codes are valid:

Return Code	Explanation
0	The Disconnect Window Service function was successful.
1	Your program is not connected to the host presentation space.
9	A system error occurred.

Notes on Using This Function

1. Once the Disconnect Window Service function is called, your application no longer manages the presentation space window.
2. Before exiting the application, request a Disconnect Window Service function for all presentation spaces that have been connected for Presentation Manager services. If the application exits with an outstanding connection for Window services, the subsystem cancels the outstanding connection.

Find Field Length (32)

PC/3270		*PC400*
DOS	Windows	
X	X	X

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

This function returns the number of characters contained in the field identified using the call PS position parameter. This includes all characters from the beginning of the target field up to the character preceding the next attribute byte.

Prerequisite Calls

Connect Presentation Space (1).

Call Parameters

Function number	Must be 32.
Data string	See table below.
Length	NA (2 is implied).
PS position	Identifies field within the host PS at which to start the Find. Can be the PS position of any byte within the field in which you desire the Find to start.

The calling 2-character data string can contain:

Code	Explanation
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.

Note: The symbol represents a required blank.

Return Parameters

This function returns a length and a return code. These lengths are valid:

Length	Explanation
= 0	When return code = 28, field length is 0. When return code = 24, host presentation space is not field-formatted.
> 0	Required field length in host presentation space.

These return codes are valid:

Return Code	Explanation
0	The Find Field Length function was successful.
1	Your program is not connected to a host session.
2	A parameter error was encountered.
7	The host presentation space position is incorrect.
9	A system error was encountered.
24	No such field was found.
28	Field length of 0 (zero) bytes.

Notes on Using This Function

Except when or T is used as the calling data string, if the field found is the same as the field from which the Find started, a return code of 24 is returned.

Find Field Position (31)

PC/3270		PC400
DOS	Windows	
X	X	X

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

Prerequisite Calls

Connect Presentation Space (1).

Call Parameters

Function number	Must be 31.
Data string	See table below.
Length	NA (2 is implied).
PS position	Identifies field within the host PS at which to start the Find. Can be the PS position of any byte within the field in which you desire the Find to start.

The calling 2-character data string can contain:

Code	Explanation
 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.

Note: The symbol represents a required blank.

Return Parameters

This function returns a length and a return code.

These lengths are valid:

Length	Explanation
= 0	When return code = 28, field length is 0. When return code = 24, host presentation space is not field-formatted.
> 0	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.

These return codes are valid:

Return Code	Explanation
0	The **Find Field Position** function was successful.
1	Your program is not connected to a host session.
2	A parameter error was encountered.
7	The host presentation space position is incorrect.
9	A system error was encountered.
24	No such field was found.
28	Field length of 0 (zero) bytes.

Notes on Using This Function

Except when or T is used as the calling data string, if the field found is the same as the field from which the Find started, a return code of 24 is returned.

Free Communications Buffer (124)

PC/3270		*PC400*
DOS	Windows	
X	X	–

This function returns to the DOS/Windows management memory a buffer that is no longer required by the application. The application should free the buffer prior to exiting the system.

Prerequisite Calls

Allocate Communications Buffer (123).

Call Parameters

Function number	Must be 124.
Data string	See the following table.
Data string length	Must be 6.
PS position	NA.

The calling data string can contain:

Byte	Definition
Position 1–2	Must be 0.
Positions 3–6	The address of the buffer.

Return Parameters

These return codes are valid:

Return Code	Explanation
0	The Free Communications Buffer function was successful.
2	An error was made in specifying parameters (PC/3270 DOS mode only).
9	A system error occurred.
41	The buffer is in use.

Notes on Using This Function

1. If the application tries to free an in-use buffer, the free request is denied. A return code of 41 is returned.
2. An application should request the Free Communications Buffer (124) function before exiting for all communication buffers that have been allocated using the Allocate Communications Buffer (123) function.
3. The Reset System (21) function will free buffers allocated by the Allocate Communications Buffer (123) function.

Get Key (51)

PC/3270		*PC400*
DOS	Windows	
X	X	X

This function lets your EHLLAPI application program retrieve a keystroke from a session specified by the Start Keystroke Intercept (50) function and either process, accept, or reject that keystroke. By placing this function in a loop, you can use it to intercept a string.

Prerequisite Calls

Start Keystroke Intercept (50).

Call Parameters

Function number	Must be 51.
Data string	See the following table.
Length	NA (8 is implied).
PS position	NA.

The calling data string can contain:

Byte	Definition
Position 1	One of the following: A 1-character presentation space short name (PSID) A blank or null indicating a function call against the host-connected presentation space.
Positions 2–8	Blanks that hold space for the symbol representation of the requested data.

Return Parameters

This function returns a data string and a return code.

These data strings are returned:

Byte	Definition
Position 1	One of the following values: A 1-character presentation space short name (PSID) A blank or null indicating a function call against the host-connected presentation space
Position 2	An option code character, one of the following characters: A for ASCII returned M for keystroke mnemonic S for special shift (Alt/Ctrl) returned with other data (PC/3270 DOS mode only)
Positions 3–8	These six bytes of the preallocated buffer space are used internally to enqueue and dequeue keystrokes. Possible combinations include: Byte 3 contains an ASCII character and byte 4 contains X'00'. Byte 3 and byte 4 contain a double-byte character. Bytes 3 contains the escape character (either @ or another character specified using the ESC=c option of function 9) and byte 4 contains a 1-byte abbreviation for a function. (See "ASCII Mnemonics" on page 32). Bytes 5 through 8 can be similar to bytes 3 and 4 if the returned ASCII mnemonic is longer than 2 bytes (for example, if the ASCII mnemonic represents Attn @A@Q, byte 5 contains @ and byte 6 contains Q). If not used, bytes 5 through 8 are set to zero (X'00').

Examples of returned data strings are provided below:

EAt E is the presentation space short name. The keystrokes are returned as ASCII (A), and the returned key is the lowercase letter t. (Bytes 4–8 = X'00').

EM@2 E is the presentation space short name. The keystrokes are returned as mnemonics, and the returned key is PF2 (Bytes 5–8 = X'00').

ES@Aa E is the presentation space short name. The keystrokes are returned in special shift state, and the received key is Alt+a.

Note: The valid shift prefixes of the special shift state are as follows:

@A — Alt-shift active

@S — Up-shift active

@r — Ctrl-shift active

The @ symbol is the default escape character. The value of the escape character can be set to any keystroke represented in ASCII by using the ESC=c option of the Set Session Parameters (9) function. If the escape character has been changed to another character using this option, the @ symbol in the following examples is replaced by the other character.

These return codes are valid:

Return Code	Explanation
0	The Get Key function was successful.
1	An incorrect presentation space was specified.
5	You specified the AID only option under the Start Keystroke Intercept (50) function, and non-AID keys are inhibited by this session type when EHLLAPI tries to write incorrect keys to the presentation space.
8	No prior Start Keystroke Intercept (50) function was called for this presentation space.
9	A system error was encountered.
20	An undefined key combination was typed.
25	The requested keystrokes are not available on the input queue.
31	Keystroke queue overflowed and keystrokes were lost.

Notes on Using This Function

1. If a return code of 31 occurs for the Get Key function, you can take one of these steps: either:

   Increase the value of the calling length parameter for the Start Keystroke Intercept (50) function.

   Run the Get Key function more frequently.

   An intercepted keystroke occupies 3 bytes in the buffer. The next intercepted keystroke is placed in the adjacent three bytes. When the Get Key function retrieves a keystroke (first in first out) the three bytes that it occupied become available for another keystroke. By increasing the size of the buffer or the rate at which keystrokes are retrieved from the buffer, you can eliminate buffer overflow.

   For the PC/3270, another way to eliminate return code 31 is to operate the PC/3270 emulator in the resume mode.

2. You can use the Send Key (3) function to pass both original keystrokes and any others that your EHLLAPI application might need to the host-connected presentation space.
3. Keystrokes arrive asynchronously and are enqueued in the keystroke queue that you have provided in your EHLLAPI application program using the Start Keystroke Intercept (50) function.
4. The Get Key function behaves like a read. When keystrokes are available, they are read into the data area that you have provided in your application.
5. For PC/3270 DOS mode, Get Key also provides a WAIT option that you can select using the Set Session Parameters (9) function. The WAIT options are interpreted slightly differently by the Get Key function. Refer to item 13 on page 154 for an explanation.

   Note: For PC/3270 Windows mode and PC400, this option works as NWAIT regardless with user's specification.

6. In the case of field support for a session, the application might be interested only in AID keys, for example, the Enter key. If so, the Start Keystroke Intercept (50) function option code should be set to D (meaning for AID Keys only).
7. To use this function, you must preallocate memory to receive the returned data string parameter. The statements required to preallocate this memory vary depending on the language in which your application is written. Refer to “Memory Preallocation” on page 17 for more information.

Get Request Completion (125)

PC/3270		*PC400*
DOS	Windows	
X	X	–

This function allows an application to determine the status of a previous asynchronous function request issued to the EHLLAPI and to obtain the function parameter list before using the data string again. This function is valid only if the user specified asynchronous (“A” or “M”) completion on a previous function call such as Read SF or Write SF.

Each asynchronous request requiring the Get Request Completion function will return a unique ID from the asynchronous request. The application must save this ID. This ID is the identification used by the Get Request Completion function to identify the desired request. The user has three request options using this function:

1. The application can query or wait on a specific asynchronous function request by supplying the Request ID of that function and a nonblank session short name.
2. The application can query or wait on the first completed asynchronous function request by supplying a Request ID of X'0000' and a blank session short name.
3. The application can query or wait on the first completed asynchronous function request for a specified session by supplying a Request ID of X'0000' and a nonblank session short name.

Prerequisite Calls

Connect Structured Fields (120) and
Allocate Communications Buffer (123) and either
Read Structured Fields (126) or
Write Structured Fields (127).

Call Parameters

Function number	Must be 125.
Data string	See the following table.
Data string length	Must be 14.
PS position	NA.

The calling data string can contain:

Byte	Definition
Position 1	A 1-character presentation space short session ID.
Position 2	N or W - N if NOWAIT is required. - W if WAIT is required.
Positions 3–4	Function request ID.
Positions 5–6	Function code of the completed asynchronous function (126 or 127). (returned)
Positions 7–10	Address of the data string of the completed asynchronous function call. (The application must not re-use the data string until the request is complete.) (returned)
Positions 11–12	Length of the data string of the completed asynchronous function call. (returned)
Positions 13–14	Return code of the completed asynchronous function call. (returned)

The Get Request Completion function behaves differently, depending on the second character of the parameter string:

N Nowait option: If a specific request ID was supplied and the function is complete, control is returned to the application with a return code of zero and a completed data string as defined below. If a Request ID of zero was supplied and any eligible asynchronous function is complete, control is returned to the application with a return code of zero and a completed data string as defined below.

W Wait option: If a specific Request ID was supplied and the function is not complete, the call waits until the function is complete before returning to the application. If the supplied Request ID was zero and no eligible asynchronous function is complete, the call waits until a function completes before returning to the calling application. On return, the return code value is zero and the data string is complete as defined below.

Return Parameters

The following return codes are valid:

Return Code	Explanation
0	The Get Request Completion function was successful.
2	An error was made in specifying parameters.
9	A system error was encountered.
38	Requested function was not complete.
42	No matching request was found.

The differences between return codes 38 and 42 are:

1. Return code 38 has three subtle differences:
   a. If a specific Request ID and session were requested, both the session and ID were found but the request is pending (not in a completed state).

b. If a zero Request ID and a specific session were requested, the specified session has pending requests but they are not satisfied (complete).

c. If a zero Request ID and a blank session were requested, pending requests were found but none were satisfied (complete).

2. Return code 42 has three subtle differences:

a. If a specific Request ID and session were requested, the specific Request ID was not found in either a pending or a completed state.

b. If a zero Request ID and a specific session were requested, the specific session contains no pending or completed requests.

c. If a zero Request ID and a blank session were requested, no pending or completed requests were found.

Notes on Using This Function

1. This function is valid only if the user specified asynchronous completion ("A" or "M" for "Asynchronous") on a previous function call such as Read SF or Write SF.

2. If the return code is zero, the application should check the returned data string for information pertaining to the completion of the requested asynchronous function.

Pause (18)

PC/3270		*PC400*
DOS	Windows	
X	X	X

This function waits for a specified amount of time. It should be used in place of *timing loops* to wait for an event to occur. A Pause function can be ended by a host event if a prior Start Host Notification (23) function has been called and the IPAUSE option is selected.

Prerequisite Calls

Connect Presentation Space (1).

Call Parameters

Function number	Must be 18.
Data string	NA.
Length	Contains the pause duration in half-second increments.
PS position	NA.

Return Parameters

The following return codes are valid:

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

Notes on Using This Function

1. Selecting the FPAUSE or IPAUSE option using the Set Session Parameters (9) function affects the length of the pause you get when you call this function. See item 6 on page 153 for more information.
2. The value entered in the calling length parameter is the maximum number of half-second intervals that the Pause function waits. For a pause of 20 seconds, a hex value of 28 (decimal 40) must be passed in the calling length parameter.
3. If you use the IPAUSE option, once a pause has been satisfied by a host event, you should call the Query Host Update (24) function to clear the queue prior to the next Pause function. The Pause function will continue to be satisfied with the pending event until the Query Host Update (24) function is complete.
4. A practical maximum value for the Pause function is 24 . Do not use the Pause function 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 EHLLAPI program execution.

   With applications requiring a high-resolution timer since the time interval created by a Pause function is approximate.

   Setting the time interval to zero in a loop.
5. IPAUSE set and the interruptible pause make an EHLLAPI application determine whether the specified host presentation space (PS) or operator information area (OIA) is updated. Three functions are used:

   Start Host Notification (23)
   Query Host Update (24)
   Stop Host Notification (25)

   By using IPAUSE when the Start function is called, an application waits until the host PS or OIA (or both) receives an update. When the receive is completed and the application can issue the Query function to determine the changes, PAUSE terminates. Then, the application issues the Search Presentation Space (6) to check whether the expected update occurs.

Post Intercept Status (52)

PC/3270		*PC400*
DOS	Windows	
X	X	X

This function informs the PC/3270 or PC400 emulator that a keystroke obtained through the Get Key (51) function was accepted or rejected. When the application rejects a keystroke, the Post Intercept Status function issues a beep.

Note: For the PC/3270, the beep function can be enabled or disabled at the alarm redefinition facility. If it is disabled, no beep occurs.

Prerequisite Calls

Start Keystroke Intercept (50).

Call Parameters

Function number Must be 52.
Data string A 2-byte data string:

Byte	Definition
Position 1	One of the following values: The 1-letter short name of the presentation space A blank or null indicating a function call against the host-connected presentation space
Position 2	One of the following characters: A for accepted keystroke R for rejected keystroke

Length NA (2 is implied).
PS position NA.

Return Parameters

These return codes are valid:

Return Code	Explanation
0	The Post Intercept Status function was successful.
1	An incorrect presentation space was specified.
2	An incorrect session option was specified.
8	No prior Start Keystroke Intercept (50) function was called for this presentation space ID.
9	A system error was encountered.

Query Close Intercept (42)

PC/3270		*PC400*
DOS	Windows	
–	X	X

This function allows the application to determine if the close option was selected from the pull-down panel.

Prerequisite Calls

Start Close Intercept (41).

Call Parameters

Function number	Must be 42.
Data string	One-character short session ID of the host presentation space.
Data string length	Must be specified (normally 1).
PS position	NA.

Return Parameters

These return codes are valid:

Return Code	Explanation
0	A close intercept event did not occur.
1	The presentation source was incorrect.
2	An error was made in specifying parameters.
8	No prior Start Close Intercept (41) function was called for this host presentation space.
9	A system error occurred.
12	The session stopped.
26	A close intercept occurred since the last query close intercept call.

Query Communications Buffer Size (122)

PC/3270		*PC400*
DOS	Windows	
X	X	–

This function allows an application to determine both the maximum and the optimum buffer size supported by the Emulation Program.

Prerequisite Calls

None.

Call Parameters

Function number	Must be 122.
Data string	See the following table.
Data string length	Must be 9.
PS position	NA.

The calling data string can contain:

Byte	Definition
Position 1	A 1-character presentation space short session ID.
Positions 2–3	16-bit field for the optimum supported inbound buffer size. (Returned value)
Positions 4–5	16-bit field for the maximum supported inbound buffer size. (Returned value)
Positions 6–7	16-bit field for the optimum supported outbound buffer size. (Returned value)
Positions 8–9	16-bit field for the maximum supported outbound buffer size. (Returned value)

Return Parameters

These return codes are valid:

Return Code	Explanation
0	The Query Communications Buffer Size function was successful.
1	A specified host presentation space short session ID was incorrect or was not connected.
2	An error was made in specifying parameters.
9	A system error occurred.
10	The function was not supported by the emulation program.

Notes on Using This Function

1. There is no way to enforce the user to use this function. It is not required since the application can be tailored to run on any system.
2. The buffer sizes returned represent the record sizes that are actually transmitted across the medium. For a DDM connection, the 8-byte header supplied in the Read and Write SF data buffer is stripped off and 1 byte containing the structured field AID value is prefixed.

   The application should compare the size of the actual data in the data buffer (which does not include the 8-byte header) to the buffer sizes returned by the Query Communications Buffer Size minus one byte. For Destination/Origin connections, the 8-byte header supplied in the Read and Write SF data buffer is stripped off and 9 bytes are then prefixed to the data. The application should compare the size of the actual data in the data buffer (which does not include the 8-byte header) to the buffer size returned by the Query Communications Buffer Size minus 9 bytes.
3. The maximum buffer sizes returned represent the maximum number of bytes supported by the PC hardware and by the Emulator. The maximum buffer size can only be used if the Host is also configured to accept at least these maximum sizes.
4. The optimum buffer sizes returned represent the optimum number of bytes supported by the both the PC hardware and the Emulator. Some network configurations might set transmission limits smaller than these values. In these cases, the data transfer buffer size override value in the Emulator configuration profile will be used for structured field support. The Query Communications Buffer Size will reflect any buffer size override values entered in the Emulator configuration profile.

Query Cursor Location (7)

PC/3270		*PC400*
DOS	Windows	
X	X	X

This function indicates the position of the cursor in the host-connected presentation space by returning the cursor position.

Prerequisite Calls

Connect Presentation Space (1).

Call Parameters

Function number	Must be 7.
Data string	NA.
Length	NA.
PS position	NA.

Return Parameters

This function returns a length and a return code.

Length	Host presentation space position of the cursor.
Return Code	The following codes are valid:

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

Query Field Attribute (14)

PC/3270		*PC400*
DOS	Windows	
X	X	X

This function returns the attribute byte of the field containing the input host PS position. This information is returned in the returned length parameter.

For the PC/3270, note also that:

The returned length parameter is set to 0 if the screen is unformatted. Attribute bytes are equal to or greater than hex C0.

Prerequisite Calls

Connect Presentation Space (1).

Call Parameters

Function number	Must be 14.
Data string	NA.
Length	NA.
PS position	Identifies the target field. Can be the PS position of any byte within the target field.

Return Parameters

This function returns a length and a return code.

Length	The attribute value, if the screen is formatted; or 0, if the screen is unformatted.
Return code	The following codes are valid:

Return Code	Explanation
0	The Query Field Attribute was successful.
1	Your program is not connected to a host session.
7	The host presentation space position is incorrect.
9	A system error was encountered.
24	Attribute byte not found or unformatted host presentation space.

Notes on Using This Function

The returned field attributes are defined in the following tables. The bit positions are in IBM format with bit 0 as the leftmost bit in the byte.

3270 field attribute:

Bit Position	Meaning
0–1	Both = 1,field attribute byte.
2	Unprotected/protected 0 = Unprotected data field 1 = Protected field.

Query Field Attribute

Bit Position	Meaning
3	A/N 0 = Alphanumeric data 1 = Numeric data only
4–5	I/SPD 00 = Normal intensity, pen not detectable 01 = Normal intensity, pen detectable 10 = High intensity, pen detectable 11 = Nondisplay, pen not detectable
6	Reserved
7	MDT 0 = Field has not been modified 1 = Field has been modified

5250 field attributes:

Bit Position	Meaning
0	Field attribute flag 0 = Non–field attribute flag 1 = Field attribute flag
1	Visibility 0 = Nondisplay 1 = Display
2	Unprotected/protected 0 = Unprotected data field 1 = Protected field
3	Intensity 0 = Normal intensity 1 = High intensity
4–6	Field type 000 = Alphanumeric data: All characters are available 001 = Alphabet only: Uppercase and lowercase, comma, period, hyphen, blank, or Dup key are available 010 = Numeric shift: Automatic shift for number 011 = Numeric data only: 0–9, comma, period, plus, minus, blank, or Dup key are available 101 = Numeric data only: 0–9, or Dup key are available 110 = Magnetic stripe reading device data only 111 = Signed-numeric data: 0–9, plus, minus, or Dup key are available
7	MDT 0 = Field has not been modified 1 = Field has been modified

Query Host Update (24)

PC/3270		PC400
DOS	Windows	
X	X	X

This function lets the program operator determine if the host has updated the host PS or OIA since:

The Start Host Notification (23) function was called (on first call to the Query Host Update function only)

The previous call to the Query Host Update function (for all calls to the Query Host Update function except the first).

Prerequisite Calls

Start Host Notification (23).

Call Parameters

Function number	Must be 24.
Data string	One-character short name of the host presentation space, or a blank or null indicating request for updates to host-connected presentation space.
Length	NA (1 is implied).
PS position	NA.

Return Parameters

The following return codes are valid:

Return Code	Explanation
0	No updates have been made since the last call.
1	An incorrect host presentation space was specified.
8	No prior Start Host Notification (23) 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	Both the OIA and the host presentation space were updated.

Notes on Using This Function

The target presentation space must be specified in the data string, even though a connection to the host presentation space is not necessary to check for updates. For Full Function DOS Mode with Non-SNA DFT connection, there might be a timing problem if this function and one of the following functions are called consecutively:

- (3) Send Key
- (5) Copy Presentation Space
- (8) Copy Presentation Space to String
- (15) Copy String to Presentation Space

– (33) Copy String to Field

– (40) Set Cursor

In such a case, call Set Session Parameters (9) with the TWAIT session option before this function, then call Wait (4) immediately after this function.

Query Session Status (22)

PC/3270		*PC400*
DOS	Windows	
X	X	X

This function is used to obtain session-specific information.

Prerequisite Calls

None.

Call Parameters

Function number — Must be 22.

Data string — An 18-byte string consisting of a 1-byte short name of the target presentation space plus 17 bytes for returned data. Position 1 can have:

A blank or a null to indicate a request against the host-connected presentation space.

An asterisk to indicate a request against the keyboard-owner presentation space.

Length — 18 bytes.

PS position — NA.

Return Parameters

This function returns a data string and a return code.

Data String: A data string of 18 bytes is returned. The bytes are defined in the following table:

Byte	Definition
Position 1	Short name (PSID).
Positions 2–9	Session Long Name (same as short name).
Position 10	Session type (F = 5250 display, G = 5250 printer).
Position 11	Session characteristics. It is expressed by a binary number including the following session characteristics bit. Bit 0 EAB 0: Session has the basic attribute. 1: Session has the extended attribute. Bit 1 PSS 0: Session does not support the programmed symbols. 1: Session supports the programmed symbols. Bits 2–7 Reserved
Positions 12–13	Number of rows in the host presentation space; expressed as a binary number.
Positions 14–15	Number of columns in the host presentation space; expressed as a binary number.
Positions 16–17	Host code page expressed as a binary number.
Position 18	Reserved.

The following return codes are valid:

Return Code	Explanation
0	The Query Session Status function was successful.
1	An incorrect host presentation space was specified.
2	An incorrect string length was made.
9	A system error was encountered.

Notes on Using This Function

1. To use this function, preallocate memory to receive the returned data string parameter. The statements required to preallocate this memory vary depending on the language in which your application is written. Refer to “Memory Preallocation” on page 17 for more information.
2. If you request to obtain the session information against the keyboard-owner presentation space (Position 1 is filled with an '*') when DOS session is active, EHLLAPI returns a blank at Position 1.

Query Sessions (10)

PC/3270		*PC400*
DOS	Windows	
X	X	X

This function returns a 12-byte data string describing each host session.

Prerequisite Calls

None.

Call Parameters

Function number — Must be 10.

Data string — Preallocated string of N * 12 bytes (N = number of sessions) or more.

Length — N * 12 bytes.
When the length is not matched to the number of sessions, the return code is 2.

PS position — NA.

Return Parameters

This function returns a data string, a length, and a return code.

Data string — The returned data string is N * 12 bytes long, where N is the number of sessions. The descriptors are concatenated into the data string and each 12-byte descriptor contains the short name, host session type, and PS size of a host session.

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

Byte	Definition
Position 1	Short name of session.
Positions 2–9	Session Long Name (same as short name).
Position 10	Session type (H=host).
Positions 11–12	Host presentation space size (this is a binary number and is not in display format). Value is 0 when Session type is H and Printer session.

Length — The number of host sessions started.

Return code The following codes are valid:

Return Code	Explanation
0	The Query Sessions function was successful. *
2	An incorrect string length was made.
9	A system error was encountered.

* For the query of the multiple sessions in the PC/3270 DOS mode, the return code is 0 when the query of at least one session is successful even if the length of the data string is incorrect.

Notes on Using This Function

1. If an application program receives RC=2 or RC=0, the number of the active sessions is returned into the length field. The application program can recognize the minimum string length by this number.
2. PC/3270 and PC400 EHLLAPI return only a data string of host sessions (Position 10, session type = H). Some 3270 and 5250 products also return a data string of PC sessions (Position 10, session type = P). Therefore, validation of session type is recommended in EHLLAPI programming.

Query System (20)

PC/3270		PC400
DOS	Windows	
X	X	X

This function can be used by an EHLLAPI application program to determine the level of PC/3270 or PC400 supports and other system-related values. This function returns a string that contains 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 under “Return Parameters.”

Prerequisite Calls

None.

Call Parameters

Function number	Must be 20.
Data string	Preallocated string of 35 bytes.
Length	NA (35 is implied).
PS position	NA.

Return Parameters

This function returns a data string and a return code.

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

Byte	Definition
Position 1	EHLLAPI version number.
Positions 2–3	EHLLAPI level number.
Positions 4–9	EHLLAPI date (month, date, year—for service purposes only).
Positions 10–12	Reserved.
Position 13	Hardware base: A= Personal Computer AT X= Personal Computer or Personal Computer XT Model 286 U= Unable to determine Z= Interrogate positions 28 and 29 for hardware base
Position 14	Program type, where P=PC/3270 or PC400.
Positions 15–16	Reserved.
Positions 17–18	PC/3270 or PC400 version as a 2-byte ASCII value.
Position 19	Reserved.
Positions 20–23	Extended Error Code 1. This is a printable ASCII string representing a hex word giving the EHLLAPI 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.*

Byte	Definition
Positions 28–29	The system model base number and submodel number in binary as follows: X'FC00' or X'FC01'=Personal Computer AT X'FC02'=Personal Computer XT X'FA01'=PS/2 Model 25 X'FA00'=PS/2 Model 30 X'FC04'=PS/2 Model 50 X'FC05'=PS/2 Model 60 X'F804' or X'F809'=PS/2 Model 70 X'F800' or X'F801'=PS/2 Model 80
Positions 30–31	NLS type expressed as a 2-byte binary number.
Position 32	One-byte printable ASCII code representing type of monitor being used as follows: A=Personal System/2 Monochrome, B=Personal System/2 Monochrome Model 30, C=CGA E=EGA G=MCGA H=Personal System/2 Model 8514 M=Monochrome V=VGA U=Unknown
Positions 33–35	Reserved.

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

Return code The following codes are valid:

Return Code	Explanation
0	The Query System function was successful; data string has been returned.
1	EHLLAPI is not loaded. (PC/3270 only)
2	An incorrect string length was made. (PC/3270 only)
9	A system error was encountered.

Notes on Using This Function

To use this function, preallocate memory to receive the returned data string parameter. The statements required to preallocate this memory vary depending on the language in which your application is written. Refer to “Memory Preallocation” on page 17 for more information.

Query Window Coordinates (103)

PC/3270		*PC400*
DOS	Windows	
–	X	X

This function requests the coordinates for the window of a presentation space. The window coordinates are returned in pels.

Note: (0,0) indicates the top-left of the coordinates.

Prerequisite Calls

Connect Window Services (101).

Call Parameters

Function number	Must be 103.
Data string	One-character short session ID of the host presentation space.
Data string length	NA (17 is implied).
PS position	NA.

The data string can contain:

Byte	Definition
Position 1	One of the following values: A 1-character presentation space short session ID. A blank or null indicating a function call against the current connection presentation space.
Positions 2–17	Reserved.

Return Parameters

This function returns a data string and a return code.

Data string — The data string can contain:

Byte	Definition	
Position 1	One of the following values: A 1-character presentation space short session ID A blank or null indicating a function call against the current connection presentation space	
Positions 2–17	Four 32-bit unsigned integers that return:	
	XLeft	Long integer in pels of left X coordinate of rectangular window relative to desktop window
	YBottom	Long integer in pels of bottom Y coordinate or rectangular window relative to desktop window
	XRight	Long integer in pels of right X coordinate of rectangular window relative to desktop window
	YTop	Long integer in pels of top Y coordinate of rectangular window relative to desktop window

Return code — The following codes are valid:

Return Code	Explanation
0	The Query Window Coordinates function was successful.
1	Your program is not connected to the host session.
9	A system error occurred.
12	The session stopped.

Read Structured Fields (126)

PC/3270		*PC400*
DOS	Windows	
X	X	–

This function allows an application to read structured field data from the host application. If the call specifies S (for synchronous), the application does not receive control until Read Structured Fields is completed. If the call specifies A. or M (for asynchronous), the application receives control immediately after the call. In either case (S, A, or M), the application provides the buffer address in which the data from the host is to be placed.

The following statements apply for a successful asynchronous completion of this function:

The return code field in the parameter list might not contain the results of the requested I/O. If the return code is not 0, then the request failed. The application must take the appropriate action based on the return code.

If the return code for this request is 0, the application must use the request ID returned with this function call to issue the Get Request Completion function call to determine the completion results of the function associated with the request ID. The Get Request Completion function call has the following information returned:

Function request ID
Address of the data string from the asynchronous request
Length of the data string
Return code of the completed function

Prerequisite Calls

Connect for Structured Fields (120) and
Allocate Communication Buffer (123).

Call Parameters

Function number	Must be 126.
Data string	See the following table.
Data string length	Length of string (8 or 14).
PS position	NA.

The calling data string can contain:

Byte	Definition
Position 1	A 1-character presentation space short session ID.
Position 2	S, A, or M. S = Synchronous. Control is not returned to the application until the read is satisfied. A = Asynchronous. Control is returned immediately to the application. M = Asynchronous. The asynchronous mode of the notification is requested. (PC/3270 Windows mode only)
Positions 3–4	2-byte destination or origin ID.

Byte	Definition
Positions 5–8	4-byte address of the buffer into which the data is to be read. The buffer must be obtained using the Allocate Communications Buffer (123) function.
Positions 9–10	Reserved.
Positions 11–12	When M is specified in position 2, the windows handle of the window that receives the message should be set. (PC/3270 Windows mode only) The message is a return value of: RegisterWindowMessage("PCSHLL") (not equal 0).
Positions 13–14	The data in these positions is ignored by EHLLAPI. However, there is no error if the migrating program has data in these positions. This data is accepted to provide compatibility with migrating applications.

Return Parameters

This function returns a data string and a return code.

Data String — If A (asynchronous) is specified in position 2 and the function is completed successfully, the following data string is returned.

Byte	Definition
Positions 9–10	2-byte function request ID. It is used by the Get Request Completion (125) function to determine the completion of this function call.

If M (asynchronous mode of notification) is specified in position 2 and the function is completed successfully, the following data string is returned.

Byte	Definition
Positions 9–10	2-byte function request ID. It is used by the Get Request Completion (125) function to determine the completion of this function call.
Positions 11–12	Task ID of asynchronous mode of notification.

If the function completes successfully, an application window receives a message. The message is a return value of RegisterWindowMessage("PCSHLL"). wParam parameter contains task ID returned by the function call. HIWORD of lParam parameter contains return code 0, which shows the function was successful, and LOWORD of lParam parameter contains function number 126.

These return codes are valid:

Return Code	Explanation
0	The Read Structured Fields function was successful.
1	A specified host presentation space short session ID was incorrect or was not connected.
2	An error was made in specifying parameters.
9	A system error occurred.
11	Resource unavailable (memory unavailable).
35	Request rejected. An outbound transmission from the host was canceled.
36	Request rejected. Lost contact with the host.
37	Successful Warning - The host is inbound disabled.

Notes on Using This Function

1. Return code 35 is returned when the first Read Structured Fields or Write Structured Fields is requested after an outbound transmission from the host is canceled. Corrective action is the responsibility of the application.
2. Return code 36 requires that the application disconnect from the Emulation Program and then reconnect to reestablish communications with the host. Corrective action is the responsibility of the application.
3. Return code 37 is returned if the host is inbound disabled. The Read Structured Fields function was successfully requested.

4. The EHLLAPI allows for a maximum of 20 asynchronous requests per application to be outstanding. A return code for unavailable resources is returned if more than 20 asynchronous requests are attempted.

The structured field data contains the application structure fields received from the host. Structured field headers are removed by the EHLLAPI before the structured field data reaches the application.

The structured field data format is as follows:

Offset	Length	Contents
0	1 word	X'0000'.
2	1 word	m (Message length: The number of bytes data in the message. The number does not include the buffer header prefix (8 bytes). This value is returned by EHLLAPI.
4	1 word	n Buffer size: The supplied length of the data buffer, which does include the 8-byte message header. This value must be set by application.
6	1 word	X'C000'.
8	1 word	Length of the first (or only) structured field message.
10	1 byte	First non–length byte of the structured field message.
		⋮
m+7	1 byte	Last byte in the structured field message.

Bytes 0–7 are the buffer header. These first 8 bytes are used by the emulation program. The user section of the buffer begins with offset 8. Bytes 8 and 9 contain the number of bytes in the first structured field (a structured field message can contain multiple structured fields), including 2 bytes for bytes 8 and 9. Bytes 8–m+7 are used for the structured field message received from the host (which can contain multiple structured fields).

The using application must furnish the complete buffer with the word at offset 0 set to zero. The buffer length must be in the word at offset 4. The word at offset 6 must be X'C000'. The emulation program will place the data message beginning at offset 8 and place the length of the message in the word at offset 2. The buffer length is not disturbed by EHLLAPI.

Receive File (91)

PC/3270		PC400
DOS	Windows	
X	X	X

This function is used to transfer a file from the host session to the personal computer session. It is used the same way as the RECEIVE command is used in the PC/3270. The Receive File function can be called by an EHLLAPI application program.

Prerequisite Calls

None.

Call Parameters

Function number — Must be 91.

Data string — Data string must consist of 128 or fewer bytes for the PC/3270 DOS mode, and 255 or fewer bytes for the PC/3270 Windows mode and PC400. Refer to the example.

Length — Length, in number of bytes, of the data string. Overridden if in EOT mode.

PS Position: — In the PC/3270 DOS mode, drive number between 1 and 8, inclusive, used to indicate the drive on which the RECEIVE.EXE file is located, where:

1 = Drive A
2 = Drive B
⋮
8 = Drive H

Note: When the Receive File function is called, the RECEIVE.EXE file must be in the directory or subdirectory that is current on the drive specified in the PS position parameter (PC/3270 DOS mode only).

In the PC/3270 Windows mode and the PC400 mode, the value must be 0.

Examples of data strings follow:

3270 Session

To receive the file from VM/CMS host system.

pc_filename [id:]fn ft [fm] [(option]

To receive the file from MVS/TSO host system

pc_filename [id:]dataset [(member)] [/password] [(option]

To receive the file from CICS host system

pc_filename [id:]host_filename [option]

5250 Session

To receive the file from AS/400 host system

pc_filename [id:]library file member [option]

Note: Parameters within [ ] can be abbreviated. Available options are listed below. For more information on the options, refer to the HELP, *Personal Communications Version 4.1 Up and Running for DOS (Full-Function Mode)*, *Personal Communications Version 4.1 Up and Running for Windows*, or *Personal Communications AS/400 and 3270 Version 4.1 Up and Running for Windows*.

Host system	Options
VM/CMS	ASCII, CRLF, APPEND, TIME n, CLEAR, NOCLEAR, PROGRESS, QUIET
MVS/TSO	ASCII, CRLF, APPEND, TIME (n), CLEAR, NOCLEAR, PROGRESS, QUIET, AVBLOCK|TRACKS|CYLINDERS
CICS	ASCII, CRLF, NOCRLF, BINARY, TIME n, CLEAR, NOCLEAR, PROGRESS, QUIET
OS/400	ASCII, CRLF, APPEND, TIME n, CLEAR, NOCLEAR, PROGRESS, QUIET

Return Parameters

This function returns the following return codes:

Return Code	Explanation
2	Parameter error or you have specified a length that is too long (more than 128 bytes in the PC/3270 DOS mode, and more than 255 bytes in the PC/3270 Windows mode and PC400) for the EHLLAPI buffer. The file transfer was unsuccessful.
3	File transfer complete.
4	File transfer complete with segmented records.
9	A system error was encountered.
27	File transfer terminated because of either a Cancel button or the time out set by the Set Session Parameter (9) function.

If you receive return code 2 or 9, there is a problem with the system or with the way you specified your data string.

For PC/3270, DOS Extended error codes reported by EHLLAPI are greater than 300. To determine the DOS Extended error code, subtract 300 and refer to the *IBM DOS Technical Reference*.

Notes on Using This Function

1. Four sets of parameters under the Set Session Parameters (9) function are related to this function. They are the STRLEN/STREOT, EOT=c, QUIET/NOQUIET and the TIMEOUT=c/TIMEOUT= session options. See items 1 and 2 on page 152, as well as items 7 and 8 on page 153 for more information.
2. If your application changes directories during execution, use of this function might require special attention. See “SEND.EXE and RECEIVE.EXE Files” on page 32.
3. This function has special memory requirements. See “Considerations for Using Send File (90) and Receive File (91) Functions (PC/3270 DOS Mode Only)” on page 31 for more information.
4. If no path is specified when the Receive File function is executed, the received file is stored in the current subdirectory, which is the directory in which your application is running.
5. For the PC/3270, this function cannot interact with a session that is not declared or with a session that is defined as a logical printer session. See the *Personal Communications Version 4.1 Reference for DOS (Full-Function Mode)*, or *Personal Communications Version 4.1 Reference for Windows*.

Release (12)

PC/3270		*PC400*
DOS	Windows	
X	X	X

This function unlocks the keyboard that is associated with the host presentation space reserved using the Reserve (11) function.

Prerequisite Calls

Connect Presentation Space (1).

Call Parameters

Function number	Must be 12.
Data string	NA.
Length	NA.
PS position	NA.

Return Parameters

The following return codes are valid:

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

Notes on Using This Function

If you do not release a host presentation space reserved by using the Reserve (11) function, you are locked out of that session until you call the Reset System (21) function, you call the Disconnect Presentation Space (2) function, or you terminate the EHLLAPI application program.

Reserve (11)

PC/3270		PC400
DOS	Windows	
X	X	X

This function locks the keyboard that is associated with the host-connected presentation space to block input from the terminal operator.

Prerequisite Calls

Connect Presentation Space (1).

Call Parameters

Function number	Must be 11.
Data string	NA.
Length	NA.
PS position	NA.

Return Parameters

The following return codes are valid:

Return Code	Explanation
0	The Reserve function was successful.
1	Your program is not connected to a host session.
5	Presentation Space cannot be used.
9	A system error was encountered.

Notes on Using This Function

1. If your EHLLAPI application program is sending a series of transactions to the host, you might need to prevent the user from accessing that session until your application processing is complete.
2. The reserved host presentation space remains locked until one of the following things occurs:

   Release (12) function is executed.
   Reset System (21) function is executed.
   Disconnect Presentation Space (2) function is executed.
   The EHLLAPI application program is terminated.
   Start Keystroke Intercept (50) function is executed in the PC/3270 Windows mode or PC400.
3. The keyboard input that a user types while it is locked by this function is enqueued and processed after the session is terminated.
4. This function locks both the mouse and the keyboard input. The application program must unlock the presentation space to enable either mouse or keyboard input.

Reset System (21)

PC/3270		PC400
DOS	Windows	
X	X	X

This function reinitializes EHLLAPI to its starting state. The session parameter options are reset to their defaults. Event notification is stopped. The reserved host session is released. The host presentation space is disconnected. Keystroke intercept is disabled.

Use the Reset System function during initialization or at program termination to reset the system to a known initial condition.

Prerequisite Calls

None.

Call Parameters

Function number	Must be 21.
Data string	NA.
Length	NA.
PS position	NA.

Return Parameters

These return codes are valid:

Return Code	Explanation
0	The Reset System function was successful.
1	EHLLAPI is not loaded.
9	A system error was encountered.

Notes on Using This Function

For PC/3270, this function can be used to check whether EHLLAPI is loaded. Place a call to this function at the start of your application and check for a return code of 1.

Search Field (30)

PC/3270		PC400
DOS	Windows	
X	X	X

This function examines a field within the connected host presentation space for the occurrence of a specified 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, in a 24-row by 80-column presentation space, 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 either protected or unprotected fields but only in a *field-formatted* host presentation space.

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

Prerequisite Calls

Connect Presentation Space (1).

Call Parameters

Function number	Must be 30.
Data string	Target string for search.
Length	Length of the target data string. Overridden in EOT mode.
PS position	Identifies the target field. For SRCHALL, can be the PS position of any byte within the target field. For SRCHFROM, is beginning point of search for SRCHFRWD or ending point of search for SRCHBKWD. See note 3 on page 137.

Return Parameters

This function returns a length and a return code.

Length — The following codes are valid:

Length	Explanation
= 0	The string was not found.
> 0	The string was found at the indicated host presentation space position.

Return code The following codes are valid:

Return Code	Explanation
0	The Search Field function was successful.
1	Your program is not connected to a host session.
2	Parameter error. Either string length of zero or EOT mode specified but EOT character was not found in calling data string.
7	The host presentation space position is not valid.
9	A system error was encountered.
24	The search string was not found, or the host PS was unformatted.

Notes on Using This Function

1. Four sets of parameters under the Set Session Parameters (9) function are related to this function. They are the SRCHALL/SRCHFROM, STRLEN/STREOT, SRCHFRWD/SRCHBKWD, and the EOT=c session options. See items 1 through 4 on page 152 for more information.
2. You can use the Set Session Parameters (9) function to determine whether your searches proceed forward (SRCHFRWD) or backward (SRCHBKWD) in a field.
3. The Search Field function normally checks the entire field (SRCHALL, default mode). However, you can use function 9 to specify SRCHFROM. In this mode, the calling PS position parameter does more than identify the target field. It also provides a beginning or ending point for the search.

   If the SRCHFRWD option is in effect, the search for the designated string begins at the specified PS position and proceeds toward the end of the field.

   If the SRCHBKWD option is in effect, the search for the designated string begins at the end of the field and proceeds backward toward the specified PS position. If the target string is not found, the search ends at the PS position specified in the calling PS position parameter.
4. If the start position of the specified search function is the second byte in a double-byte character, the search is started from the next character for SRCHFRWD and from the character for SRCHBKWD. If the last character of the specified string is the first byte of a double-byte character, the character is not searched.
5. The search ignores a pair of SO and SI in the presentation space. When you search a double-byte control character, put SO (X'0E') before the character and SI (X'0F') after it. For example, X'0E000C0F' in the data string is treated as a double-byte character FF (X'000C').

Search Presentation Space (6)

PC/3270		PC400
DOS	Windows	
X	X	X

This function lets your EHLLAPI program examine the host presentation space for a specified string.

Prerequisite Calls

Connect Presentation Space (1).

Call Parameters

Function number	Must be 6.
Data string	Target string of search.
Length	Length of the target data string. Overridden if in EOT mode.
PS position	Position within the host presentation space where the search is to begin (SRCHFRWD option) or to end (SRCHBKWD option). Overridden in SRCHALL (default) mode.

Return Parameters

This function returns a length and a return code.

Length The following codes are valid:

Length	Explanation
= 0	The string was not found.
> 0	The string was found at the indicated host presentation space position.

Return code The following codes are valid:

Return Code	Explanation
0	The Search Presentation Space function was successful.
1	Your program is not connected to a host session.
2	An error was made in specifying parameters
7	The host presentation space position is incorrect.
9	A system error was encountered.
24	The search string was not found.

Notes on Using This Function

1. Four sets of parameters under the Set Session Parameters (9) function are related to this function. They are the SRCHALL/SRCHFROM, STRLEN/STREOT, SRCHFRWD/SRCHBKWD, and the EOT=c session options. See items 1 through 4 on page 152 for more information.
2. You can use the Set Session Parameters (9) function to specify SRCHBKWD. When this option is in effect, the search operation locates the *last* occurrence of the string.
3. The Search Presentation Space function normally checks the entire host presentation space. However, you can use the Set Session Parameters (9) function to specify SRCHFROM. In this mode, the calling PS position parameter specifies a beginning or ending point for the search.

   If the SRCHFRWD option is in effect, the search for the designated string begins at the specified PS position and proceeds toward the end of the host presentation space.

   If the SRCHBKWD option is in effect, the search for the designated string begins at the end of the PS and proceeds backward toward the specified PS position. If the target string is not found, the search ends at the PS position specified in the calling PS position parameter.
4. The SRCHFROM option is useful if you are looking for a keyword that might occur more than once in the host presentation space.
5. The Search Presentation Space function is useful in determining when the host presentation space is available. If your EHLLAPI application is expecting a specific prompt or message before sending data, the Search Presentation Space function allows you to check for a prompt message before continuing.
6. If the start position of the specified search function is the second byte in a double-byte character, the search is started from the next character for SRCHFRWD and from the character for SRCHBKWD. If the last character of the specified string is the first byte of a double-byte character, the character is not searched.
7. The search ignores a pair of SO and SI in the presentation space. When you search a double-byte control character, put SO (X'0E') before the character and SI (X'0F') after it. For example, X'0E000C0F' in the data string is treated as a double-byte character FF (X'000C').

Send File (90)

PC/3270		PC400
DOS	Windows	
X	X	X

This function is used to transfer a file from the personal computer session where EHLLAPI is running to a host session.

Prerequisite Calls

None.

Call Parameters

Function number	Must be 90.
Data string	Data string must consist of 128 or fewer bytes for the PC/3270 DOS mode, and 255 or fewer bytes for the PC/3270 Windows mode and PC400. Refer to the example.
Length	Length of the data string. Overridden if in EOT mode.
PS position	In the PC/3270 DOS mode, drive number between 1 and 8, inclusive, used to indicate the drive on which the RECEIVE.EXE file is located, where: 1 = Drive A 2 = Drive B ⋮ 8 = Drive H When the Receive File function is called, the RECEIVE.EXE file must be in the directory or subdirectory that is current on the drive specified in the PS position parameter (PC/3270 DOS mode only). In the PC/3270 Windows mode and the PC400, it must be 0.

Following are examples of data strings.

3270 Session

To send the file from VM/CMS host system

```
pc_filename [id:]fn ft [fm] [(option]
```

To send the file from MVS/TSO host system

```
pc_filename [id:]dataset [(member)] [/password] [(option]
```

To send the file from CICS host system

```
pc_filename [id:]host_filename [option]
```

5250 Session

To send the file from AS/400 host system

```
pc_filename [id:]library file member [option]
```

Note: Parameters within [ ] can be abbreviated. Available options are listed below. For more information of the options, refer to the HELP, *Personal Communications Version 4.1 Up and Running for DOS (Full-Function Mode)*, *Personal Communications Version 4.1 Up and Running for*

Windows, or *Personal Communications AS/400 and 3270 Version 4.1 Up and Running for Windows*.

Host system	Options
VM/CMS	ASCII, CRLF, APPEND, LRECL n, RECFM v|f, TIME n, CLEAR, NOCLEAR, PROGRESS, QUIET
MVS/TSO	ASCII, CRLF, APPEND, LRECL (n), RECFM (v|f|u), TIME (n), CLEAR, NOCLEAR, PROGRESS, QUIET, BLKSIZE (n), SPACE (n[,m]), AVBLOCK|TRACKS|CYLINDERS
CICS	ASCII, CRLF, BINARY, TIME n, CLEAR, NOCLEAR, PROGRESS, QUIET
OS/400	ASCII, CRLF, APPEND, SRC, LRECL n, TIME n, CLEAR, NOCLEAR, PROGRESS, QUIET

Return Parameters

This function returns these codes:

Return Code	Explanation
2	Parameter error or you have specified a length that is too long (more than 128 bytes in the PC/3270 DOS mode; more than 255 bytes in the PC/3270 Windows mode and PC400) for the EHLLAPI buffer. The file transfer was unsuccessful.
3	File transfer complete.
4	File transfer complete with segmented records.
5	Workstation file name is incorrect or not found. File transfer was canceled.
9	A system error was encountered.
27	File transfer terminated because of either a Cancel button or the time out set by the Set Session Parameter (9) function.

If you receive return code 2 or 9, there is a problem with the system or with how you specified your data string.

For PC/3270, DOS Extended error codes reported by EHLLAPI are greater than 300. To determine the DOS Extended error code, subtract 300 and refer to the *IBM DOS Technical Reference*.

Notes on Using This Function

1. Four sets of parameters under the Set Session Parameters (9) function are related to this function. They are the QUIET/NOQUIET, STRLEN/STREOT, TIMEOUT=c/TIMEOUT= , and the EOT=c session options. See items 1 and 2 on page 152, as well as items 7 and 8 on page 153 for more information.
2. If your application changes directories during execution, use of this function might require special attention. See "SEND.EXE and RECEIVE.EXE Files" on page 32.
3. This function has special memory requirements. See "Considerations for Using Send File (90) and Receive File (91) Functions (PC/3270 DOS Mode Only)" on page 31 for more information.
4. If no path is specified when the Receive File function is executed, the received file is stored in the current subdirectory, the directory in which your application is running.
5. For the PC/3270, this function cannot interact with a session that is not declared or with a session that is defined as a logical printer session. See the *Personal Communications Version 4.1 Reference for DOS (Full-Function Mode)*, or *Personal Communications Version 4.1 Reference for Windows*.

Send Key (3)

PC/3270		PC400
DOS	Windows	
X	X	X

This function is used to send either a keystroke or a string of keystrokes to the host presentation space.

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

Prerequisite Calls

Connect Presentation Space (1).

Call Parameters

Function number	Must be 3.
Data string	A string of keystrokes, maximum 255. Upper and lower case ASCII characters are represented literally. Function keys and shifted function keys are represented by mnemonics. See “Keyboard Mnemonics” on page 145.
Length	Length of the source data string. Overridden if in EOT mode.
PS position	NA.

Return Parameters

These return codes are valid:

Return Code	Explanation
0	The keystrokes were sent; status is normal.
1	Your program is not connected to a host session.
2	An incorrect parameter was passed to EHLLAPI.
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

1. Four sets of parameters under the Set Session Parameters (9) function are related to this function. They are the AUTORESET/NORESET, STRLEN/STREOT, EOT=c and the ESC=c session options. See items 1 and 2 on page 152 plus items 9 and 10 on page 154 for more information.
2. Keystrokes cannot be sent to the host session when the keyboard is locked or busy. You can check this condition with the Wait (4) function.
3. If the host is busy, input might be rejected.
4. The data string length must be explicitly defined by the default length parameter, but it can be defined implicitly by EOT=c option of the Set Session Parameters (9) function.

When explicitly defining length (see item 1), the value for the length parameter passed by the application must be calculated. For this calculation, allow 2 bytes for compound keystrokes such as @E and allow 4 bytes for compound keystrokes such as @A@C.

5. To send special control keys, a compound character coding scheme is used. In this coding scheme, one keystroke is represented by a sequence of two to four ASCII characters. The first and third character are always the escape character. The second and fourth characters are always keycodes.

   To send 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 145 .

   This compound coding technique allows an ASCII string representation of all necessary keystroke codes without requiring the use of complex hexadecimal key codes.

   The default escape character is @. The value of the escape character can be changed to any other character with the ESC=c option of the Set Session Parameters (9) function.

6. Users needing higher levels of performance should use the Copy String to Field (33) or Copy String to Presentation Space (15) function rather than send keystrokes with the Send Key (3) function. But remember, only the Send Key (3) function can send the special control keys.

7. Refer to "Set Session Parameters (9)" Session option 10 on page 154 (NORESET option) to improve the performance of this function.

   Unless NORESET is required, the reset mnemonic is added to the keystroke strings as a prefix. Therefore, all resettable status except input inhibit are reset.

   NORESET option is not the same as the Reset System (21) function.

8. The keystroke strings, including the AID key, are sent to the host via multiple paths. Each path sends the strings before the first AID key (or including the AID key). EHLLAPI adjusts the string length and the start position of each path. For a host application program, any keystroke might be lost by the AID key process. Therefore, do not send a keystroke list that includes plural AID keys.

9. During @P (Print) or @A@T (Print Presentation Space) process, all requests that update the presentation space are rejected. If the presentation space is busy or the interruption request occurs during the print request, the mnemonic @A@R (Device Reset–Cancel to print the Presentation Space) cancels the request and resets the status.

10. For compatibility with OS/2 EHLLAPI, the mnemonics of "Page Up" and "Page Down" key are changed in V4.1. If you want to set those mnemonics to the same ones as V4.0, add the following statements to the PCSWIN.INI file in your PRIVATE directory:

    ```
    [API]
    OldPageUpDown=Y
    ```

Keyboard Mnemonics

The keyboard mnemonics provide the ASCII characters representing the special function keys of the keyboard in the personal computer. The abbreviation codes make the mnemonics for special keys easy to remember. An alphabetic key code is used for the most common key. For example, Clear key is *C*, and Tab key is *T*. Note that the uppercase and lowercase alphabetic characters are mnemonic abbreviations for different keys.

The following table shows the mnemonics using alphabetic uppercase.

Table 5. Mnemonics Using Alphabetic Uppercase

ASCII Mnemonic	Meaning	PC/3270	PC400
@B	Backtab	X	X
@C	Clear	X	X
@D	Delete	X	X
@E	Enter	X	X
@F	Erase EOF	X	X
@H	Help	–	X
@I	Insert	X	X
@J	Jump (Set focus in PM)	X	–
@L	Cursor left	X	X
@N	New line	X	X
@O	Space	–	X
@P	Print	X	X
@R	Reset	X	X
@T	Tab	X	X
@U	Cursor up	X	X
@V	Cursor down	X	X
@Y	Caps lock (No operation)	X	–
@Z	Cursor right	X	X

The following table shows the mnemonics using a number or alphabetic lowercase.

Table 6 (Page 1 of 2). Mnemonics Using Number or Alphabetic Lowercase

ASCII Mnemonic	Meaning	PC/3270	PC400
@0	Home	X	X
@1	PF1/f1	X	X
@2	PF2/f2	X	X
@3	PF3/f3	X	X
@4	PF4/f4	X	X
@5	PF5/f5	X	X
@6	PF6/f6	X	X
@7	PF7/f7	X	X
@8	PF8/f8	X	X
@9	PF9/f9	X	X
@a	PF10/f10	X	X

Table 6 (Page 2 of 2). Mnemonics Using Number or Alphabetic Lowercase

ASCII Mnemonic	Meaning	PC/3270	PC400
@b	PF11/f11	X	X
@c	PF12/f12	X	X
@d	PF13	X	X
@e	PF14	X	X
@f	PF15	X	X
@g	PF16	X	X
@h	PF17	X	X
@i	PF18	X	X
@j	PF19	X	X
@k	PF20	X	X
@l	PF21	X	X
@m	PF22	X	X
@n	PF23	X	X
@o	PF24	X	X
@s	Scrlk (No operation)	X	–
@t	Num lock (No operation)	X	–
@u	Page Up	–	X
@v	Page Down	–	X
@x	PA1	X	–
@y	PA2	X	–
@z	PA3	X	–

The following table shows the mnemonics using the combination @A and @alphabetic uppercase (A–Z) key.

Table 7. Mnemonics Using @A and Alphabetic Uppercase

ASCII Mnemonic	Meaning	PC/3270	PC400
@A@C	Test	–	X
@A@D	Erase word	X	X
@A@E	Field exit	–	X
@A@F	Erase input	X	X
@A@H	System request	X	X
@A@I	Change insert mode	–	X
@A@J	Cursor selection	X	–
@A@L	Cursor left fast	X	X
@A@Q	Attention	X	X
@A@R	Device cancel (Cancel to print presentation space)	X	–
@A@T	Print presentation space	X	–
@A@U	Cursor up fast	–	X
@A@V	Cursor down fast	–	X
@A@Z	Cursor right fast	X	X

The following table shows the mnemonics using the combination @A and @number or @A and @alphabetic lowercase (a–z) key.

Table 8. Mnemonics Using @A and @Number or @A and @Alphabetic Lowercase

ASCII Mnemonic	Meaning	PC/3270	PC400
@A@9	Reverse video	X	–
@A@a	Backspace erase	X	–
@A@b	Underscore	X	–
@A@c	Reset reverse video	X	–
@A@d	Red	X	–
@A@e	Pink	X	–
@A@f	Green	X	–
@A@g	Yellow	X	–
@A@h	Blue	X	–
@A@i	Turquoise	X	–
@A@j	White	X	–
@A@l	Reset host colors	X	–
@A@n	Jump to session 1	–	X
@A@o	Jump to session 2	–	X
@A@p	Jump to session 3	–	X
@A@q	Jump to session 4	–	X
@A@r	Jump to session 5	–	X
@A@t	Print (personal computer)	–	X
@A@y	Forward word tab	X	X
@A@z	Backward word tab	X	X

The following table shows the mnemonics using the combination @A and @special character.

Table 9. Mnemonics Using @A and @Special Character

ASCII Mnemonic	Meaning	PC/3270	PC400
@A@–	Field –	–	X
@A@+	Field +	–	X
@A@<	Backward record	–	X

The following table shows the mnemonics using the combination @S and @alphabetic lowercase.

Table 10. Mnemonics Using @S and @Alphabetic Lowercase

ASCII Mnemonic	Meaning	PC/3270	PC400
@S@x	Dup	X	X
@S@y	Field mark	X	–

The following table shows the mnemonics using the combination @X and @number or @alphabetic lowercase (a–z).

Table 11. Mnemonics Using @S and @Alphabetic Lowercase

ASCII mnemonic	Meaning	PC/3270	PC400
@X@1	Display SO/SI	X	X
@X@5	Generate SO/SI	–	X
@X@6	Display attribute	–	X
@X@7	Forward character	–	X
@X@c	Split vertical bar (¦)	–	X

The following character keys are interpreted as they are.

Table 12. Character Keys

ASCII Mnemonic	Meaning	PC/3270	PC400
a–z	a–z	X	X
A–Z	A–Z	X	X
0–9	0–9	X	X
~	~	X	X
#	#	X	X
!	!	X	X
$	$	X	X
%	%	X	X
&	&	X	X
,	,	X	X
(	(	X	X
)	)	X	X
*	*	X	X
+	+	X	X
'	'	X	X
-	-	X	X
.	.	X	X
/	/	X	X
:	:	X	X
;	;	X	X
<	<	X	X
>	>	X	X
=	=	X	X
?	?	X	X
{	{	X	X
}	}	X	X
[	[	X	X
]	]	X	X
|	|	X	X

The following table shows mnemonics using special characters.

Table 13. Mnemonics Using Special Character Keys

ASCII Mnemonic	Meaning	PC/3270	PC400
@@	@	X	X
@$	Alternate cursor	X	–
@<	Backspace	X	X

Set Cursor (40)

PC/3270		*PC400*
DOS	Windows	
X	X	X

This function is used to set the position of the cursor within the host presentation space. Before using the Set Cursor function, a personal computer application must be connected to the host presentation space.

Prerequisite Calls

Connect Presentation Space (1).

Call Parameters

Function number	Must be 40.
Data string	NA.
Length	NA.
PS position	Desired cursor position in the connected host presentation space.

Return Parameters

These return codes are valid:

Return Code	Explanation
0	The cursor was successfully located at the specified position.
1	Your program is not connected to a host session.
4	The session is busy.
7	A cursor location <1 or > the size of the connected host presentation space was specified.
9	A system error occurred.

Notes on Using This Function

If the specified cursor is the second byte of the double-byte character, the cursor moves to the first byte of the character and the error is not returned.

Set Session Parameters (9)

PC/3270		*PC400*
DOS	Windows	
X	X	X

This function lets you change certain default session options in EHLLAPI. When EHLLAPI is loaded, the default settings for session options are as indicated by the underscored entries in the tables that appear beginning on page 152. Any, some, or all of these settings can be changed by including the desired option in the calling data string as explained below. Specified settings remain in effect until:

Changed by a subsequent Set Session Parameters (9) function that specifies a new value.

The Reset System (21) function is executed.

The EHLLAPI application program is terminated.

The following table lists those EHLLAPI functions that are affected by session options. Functions not listed in the table are not affected by any of the session options. Session options that affect each function are indicated by corresponding entries in the "See Items" column. These entries are indexed to the list that follows "Call Parameters" on page 152.

Function Number	Function Name	See Items
1	Connect Presentation Space	11,16
2	Disconnect Presentation Space	11
3	Send Key	1, 2, 9, 10
4	Wait	13
5	Copy Presentation Space	5, 14, 15, 17, 18
6	Search Presentation Space	1, 2, 3, 4
8	Copy Presentation Space to String	5, 14, 15, 17, 18
15	Copy String to Presentation Space	1, 2
18	Pause	6
30	Search Field	1, 2, 3, 4
33	Copy String to Field	1, 2
34	Copy Field to String	5, 14, 15, 18
51	Get Key	9, 13
90	Send File	1, 2, 7, 8
91	Receive File	1, 2, 7, 8

Prerequisite Calls

None.

Call Parameters

Function number	Must be 9.
Data string	String containing the desired values of those session options that are to be changed. The data string can contain any of the values in the tables of "Session Options." The values should be placed on the data string line, separated by commas or blanks. The sets of parameters are explained in terms of the functions they affect.
Length	Explicit length of the source data string (STREOT option is not allowed).
PS position	NA.

Session Options

The following tables show the session options. The default is underlined.

1. The values in the following table determine how the data string length is defined for functions Send Key (3), Search Presentation Space (6), Copy String to Presentation Space (15), Search Field (30), Copy String to Field (33), Send File (90), and Receive File (91).

Value	Explanation
_ STRLEN	An explicit length is passed for all strings.
STREOT	Lengths are not explicitly coded. Calling (source) data strings are terminated with an EOT character.

2. The statement in the following table is used to specify the character that is used as the end-of-text (EOT) delimiter in the calling (source) data string for EHLLAPI functions Send Key (3), Search Presentation Space (6), Copy String to Presentation Space (15), Search Field (30), Copy String to Field (33), Send File (90), and Receive File (91).

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

   To be valid, c must be entered as a 1-byte string literal character with no preceding blanks. The EOT character specified by this statement is used to determine the length of a calling data string only when the STREOT option (see item 1) is in effect.

3. The values in the following table affect the Search Presentation Space (6) and Search Field (30) search functions.

Value	Explanation
_ SRCHALL	The Search Presentation Space (6) function and Search Field (30) function scan the entire host presentation space or field.
SRCHFROM	The Search Presentation Space (6) function and Search Field (30) function start from a specified PS position (for SRCHFRWD) or ends at a specified PS position (for SRCHBKWD).

4. The values in the following table affect the Search Presentation Space (6) and Search Field (30) search functions. They determine the direction for the search.

Value	Explanation
_ SRCHFRWD	The Search Presentation Space (6) function and Search Field (30) function perform in an ascending direction.
SRCHBKWD	The Search Presentation Space (6) function and Search Field (30) function perform in a descending direction. A search is satisfied if the first character of the requested string starts within the bounds specified for the search.

5. The values in the following table determine how attribute bytes are treated for functions Copy Presentation Space (5), Copy Presentation Space to String (8) and Copy Field to String (34).

Value	Explanation
_ NOATTRB	Convert all unknown values to blanks.
ATTRB	Pass back all codes that do not have an ASCII equivalent as their original values.
NULLATTRB	Copies null characters at field attribute positions (PC/3270 Windows Mode only)

6. The values in the following table affect the Pause (18) function.

Value	Explanation
_ FPAUSE	If full-duration pause, pause lasts for the length of time you specified in the Pause (18) function.
IPAUSE	Interruptible pause. After the Start Host Notification (23) function is executed, a host event satisfies a Pause.

7. The values in the following table determine whether messages generated by file transfer functions Send File (90) and Receive File (91) are displayed.

Value	Explanation
_ NOQUIET	SEND and RECEIVE messages are displayed.
QUIET	SEND and RECEIVE messages are not displayed.

8. The statements in the following table determine how long PC/3270 or PC400 EHLLAPI waits before it automatically issues a Cancel during execution of file transfer functions Send File (90) and Receive File (91). To be valid, c must be an Arabic number 0–9 or a capital letter J–N and must not be preceded by a blank.

Value	Explanation
_ TIMEOUT=0	A Cancel is automatically issued following a 20-second (approximate) delay.
TIMEOUT=c	A Cancel is automatically issued following a specified delay. A 1-character indicator from the table below tells PC/3270 or PC400 how many 30-second cycles it should accept before issuing a Cancel itself.

Character	Value (in minutes)	Character	Value (in minutes)
1	0.5	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

9. The statement in the following table is used to define the escape character for keystroke mnemonics. This session option affects functions Send Key (3) and

Get Key (51). The value of c must be entered as a 1-byte literal character string with no preceding blanks.

Value	Explanation
ESC=c	Specifies the escape character for keystroke mnemonics (@ is the default). Do not leave a blank after the equal sign. A blank is not a valid escape character.

10. The values in the following table determine whether EHLLAPI automatically precedes strings sent using the Send Key (3) function with a reset.

Value	Explanation
_ AUTORESET	EHLLAPI tries to reset all inhibited conditions by prefixing all strings of keys sent using the Send Key (3) function with a reset.
NORESET	Do not AUTORESET.

11. The values in the following table affect the manner in which function Connect Presentation Space (1) function and Disconnect Presentation Space (2) function are performed in the PC/3270 DOS mode.

Value	Explanation
_ CONLOG	Establishes a logical connection between personal computer session and host session. During Connect, does not jump to the requested presentation space. During Disconnect, does not jump back to DOS session.
CONPHYS	Establishes a physical connection between the personal computer session and a host session. During Connect, jumps to requested presentation space. During Disconnect, jumps back to DOS session. (PC/3270 DOS mode only)

12. The following values enable or disable the EHLLAPI trace facility that is used to debug your EHLLAPI program in the PC/3270 DOS mode. This session option does not affect the execution of any EHLLAPI function other than performance. It simply determines whether the execution of a specified function is traced. For more information about trace messages, refer to “Debugging” on page 23.

Value	Explanation
_ TROFF	Turns trace off.
TRON	Turns trace on. When TRON is in effect, all executed EHLLAPI functions are traced except the Reset System (21) function. (PC/3270 DOS mode only)

Note: This option works as TROFF regardless of user's specification.

13. The values in the following table affect the Wait (4) function and Get Key (51) function. For each value, there are two different effects, one for each function.

Value	Explanation
_ TWAIT	For Wait (4) function, waits up to a minute before timing out on XCLOCK (X []) or XSYSTEM. For Get Key (51) function, does not return control to your EHLLAPI application program until it has intercepted a key (normal or AID key based on the option specified under the Start Keystroke Intercept (50) function).

Value	Explanation
LWAIT	For Wait (4) function, waits until XCLOCK (X [])/XSYSTEM clears. This option is not recommended since control does not return to your application until the host is available. For Get Key (51) function, does not return control to your EHLLAPI application program until it has intercepted a key (normal or AID key based on the option specified under the Start Keystroke Intercept (50) function).
NWAIT	For Wait (4) function, checks status and returns immediately (no wait). For Get Key (51) function, returns return code 25 (keystrokes not available) in the fourth parameter if nothing is queued matching the option specified under the Start Keystroke Intercept (50) function.

Note: Using of NWAIT is recommended.

14. The items in the following table affect functions Copy Presentation Space (5), Copy Presentation Space to String (8), and Copy Field to String (34). Extended attribute bytes (EAB) include extended character attributes and extended field attributes.

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

15. The items in the following table affect functions Copy Presentation Space (5), Copy Presentation Space to String (8), and Copy Field to String (34).

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

16. The items in the following table affect the Connect Presentation Space (1) function in the PC/3270 DOS mode. They are available if CONPHYS is specified.

Value	Explanation
_ NORESUME	Suspends DOS session while host session is active. (PC/3270 DOS mode only)
RESUME	Resumes DOS session while host session is active. (PC/3270 DOS mode only)

17. The items in the following table affect functions Copy Presentation Space (5) and Copy Presentation Space to String (8) if NOATTRIB and NOEAB are specified in the PC/3270.

Value	Explanation
_ NOBLANK	Convert all unknown values to X'00'; PC/3270 DOS mode default.
BLANK	Convert all unknown values to X'20'; PC/3270 Windows mode default.

In PC/3270 Windows mode, default value is BLANK. If you want to change the default value to NOBLANK, add the following statement in the PCSWIN.INI file (SYS:¥PCOMWIN¥PRIVATE¥PCSWIN.INI).

```
[API]
NullToBlank=NO
```

18. The values in the following table affect the copy functions COPY PRESENTATION SPACE (5), COPY PRESENTATION SPACE TO STRING (8), and COPY FIELD TO STRING (34).

Value	Explanation
DISPLAY	Copy nondisplay fields in the presentation space to the target buffer area in the same manner as display fields. Current applications function normally (PC/3270 Windows mode only).
NONDISPLAY	Do not copy nondisplay fields in the presentation space to the target buffer as a string of null characters. This allows applications to display the copied buffers in the the presentation window without displaying confidential information, such as passwords (PC/3270 Windows mode only).

Return Parameters

This function returns a length and a return code.

Length — Number of valid session parameters that are set.
Return code — The following codes are valid:

Return Code	Explanation
0	The session parameters have been set.
2	One or more parameters were incorrect.
9	A system error was encountered.

Start Close Intercept (41)

PC/3270		PC400
DOS	Windows	
–	X	X

This function allows the application to intercept close requests generated when a user selects the close option from the emulator session window. This function intercepts the close request and discards it until a Stop Close Intercept (43) function is requested.

After using this function, your application program can use the Query Close Intercept (42) function to determine when a close request has occurred.

Prerequisite Calls

None.

Call Parameters

Function number	Must be 41.
Data string	Preallocated 5-byte string. Refer to the following table.
Data string length	The length must be specified (normally 5 or 6 (asynchronous mode)).
PS position	NA.

The data string contains the following items.

Byte	Definition
Position 1	A host presentation space short name (PSID)
Positions 2–3	When M is specified in position 6, the windows handle of the window that receives the message should be set. (PC/3270 Windows mode only) The message is a return value of: RegisterWindowMessage("PCSHLL") (not equal 0).
Positions 4–5	The data in these positions is ignored by EHLLAPI. However, no error is caused if the migrating program has data in these positions. This data is accepted to provide compatibility with migrating applications.
Position 6	If you request the asynchronous mode of the notification, specify M. (PC/3270 Windows mode only)

Return Parameters

This function returns a data string and a return code.

Data string — If M (asynchronous mode of notification) is specified in position 2 and the function is completed successfully, the following data string is returned.

Byte	Definition
Positions 2–3	Task ID of asynchronous mode of notification.

If a user selects the close option, an application window receives the message. The message is a return value of RegisterWindowMessage("PCSHLL"). wParam parameter contains the task ID returned by the function call. HIWORD of lParam

parameter contains Return Code 26, which shows a close intercept occurred, and LOWORD of lParam parameter contains function number 41.

These return codes are valid:

Return Code	Explanation
0	The Start Close Intercept function was successful.
1	An incorrect host presentation space was specified.
2	A parameter error occurred.
9	A system error occurred.
10	The function was not supported by the emulation program.

Note: This is not an exclusive call. Multiple applications can request this function for the same short session ID.

If there are no applications intercepting close requests for a logical terminal, any subsequent close requests selected by the user from the emulator operations dialog result in a normal stop requested for that logical terminal.

Start Host Notification (23)

PC/3270		*PC400*
DOS	Windows	
X	X	X

This function begins the process by which your EHLLAPI application program determines if the host PS or OIA have been updated.

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

Prerequisite Calls

None.

Call Parameters

Function number	Must be 23.
Data string	Preallocated string. See the note below.
Length	NA (6 or 7 is implied).
PS position	NA.

Note: The calling data string contains these elements:

Byte	Definition
Position 1	One of the following values: A specific host presentation space short name (PSID) A blank or null indicating a request against the host-connected host presentation space
Position 2	One of the following characters: The character B asking for notification of both host presentation space and OIA updates The character O asking for notification of only OIA updates The character P asking for notification of only host presentation updates The character M requesting the asynchronous mode of the notification. (PC/3270 Windows mode only) When M is specified, an event selection character, P, O, or B must be placed in position 7.
Positions 3–4	When M is specified in position 2, the windows handle of the window that receives the message should be set. (PC/3270 Windows mode only) The message is a return value of: RegisterWindowMessage("PCSHLL") (not equal 0).
Positions 5–6	The data in these positions is ignored by EHLLAPI. However, no error is caused if the migrating program has data in these positions. This data is accepted to provide compatibility with migrating applications.
Position 7	Replace with one of the following characters if position 2 is M (PC/3270 Windows mode only): The character B asking for notification of both host presentation space and OIA updates The character O asking for notification of only OIA updates The character P asking for notification of only host presentation updates

Return Parameters

This function returns a data string and a return code.

Data string — If M (asynchronous mode of notification) is specified in position 2 and the function is completed successfully, the following data string is returned.

Byte	Definition
Positions 3–4	Task ID of asynchronous mode of notification.

If OIA or presentation space is updated, an application window receives the message. The message is a return value of RegisterWindowMessage("PCSHLL"). wParam parameter contains the task ID returned by the function call. HIWORD of lParam parameter contains return code 21 (shows the OIA is updated), 22 (shows the host presentation space is updated), or 23 (shows both the OIA and the host presentation space are updated), and LOWORD of lParam parameter contains function number 23.

These return codes are valid:

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

Notes on Using This Function

1. For the PC/3270, this function cannot interact with a session that is not declared or with a session that is defined as a logical printer session. See the *Personal Communications Version 4.1 Up and Running for DOS (Full-Function Mode)*, the *Personal Communications Version 4.1 Reference for DOS (Full-Function Mode)*, the *Personal Communications Version 4.1 Up and Running for Windows*, or the *Personal Communications Version 4.1 Reference for Windows*.
2. An application program can issue this function in some host sessions. The Pause (18) function can notify the application when one or more host sessions (PS, OIA, or both of them) are updated. The Query Host Update (24) function can be used to determine whether a PS, OIA, or both have been updated.
3. The next Start Host Notification function is not valid while the option is the same as one of the first Start Host Notification function.
4. This is not an exclusive call. Multiple applications can request this function for the same short session ID.

Start Keystroke Intercept (50)

PC/3270		*PC400*
DOS	Windows	
X	X	X

This function allows a PC application to filter any keystrokes sent to a session by a terminal operator. After a call to this function, keystrokes are intercepted and saved until the keystroke queue overflows or until the Stop Keystroke Intercept (53) function or Reset System (21) function is called. The intercepted keystrokes can be:

Received through the Get Key (51) function and sent to the same or another session with the Send Key (3) function

Accepted or rejected through the Post Intercept Status (52) function

Replaced by other keystrokes with the Send Key (3) function

Used to trigger other processes

Prerequisite Calls

None.

Call Parameters

Function number	Must be 50.
Data string	See the following table.
Length	Number of bytes in the keystroke intercept buffer, 256 recommended. EHLLAPI allocates 8 bytes minimum for this buffer.
PS position	NA.

Byte	Definition
Position 1	One of the following values: A specific host presentation space short name (PSID) A blank or null indicating a request against the host-connected host presentation space
Position 2	An **option code** character: D for AID keystrokes only L for all keystrokes M for requesting the asynchronous mode of the notification. (PC/3270 Windows mode only) When M is specified, a code character, D or L must be placed in position 7.
Positions 3–4	When M is specified in position 2, the windows handle of the window that receives the message should be set. (PC/3270 Windows mode only) The message is a return value of RegisterWindowMessage("PCSHLL") (not equal 0).
Positions 5–6	The data in these positions is ignored by EHLLAPI. However, no error is caused if the migrating program has data in these positions. This data is accepted to provide compatibility with migrating applications.
Position 7	Replace with one of the following characters if position 2 is M (PC/3270 Windows mode only): D for AID keystrokes only L for all keystrokes

Return Parameters

This function returns a data string and a return code.

Data string — If M (asynchronous mode of notification) is specified in position 2 and the function is completed successfully, the following data string is returned.

Byte	Definition
Positions 3–4	Task ID of asynchronous mode of notification.

If a user sends keystrokes to a session, an application window receives the message. The message is a return value of RegisterWindowMessage("PCSHLL"). wParam parameter contains Task ID returned by the function call. HIWORD of lParam parameter contains return code 0, which shows function was successful, and LOWORD of lParam parameter contains function number 50.

These return codes are valid:

Return Code	Explanation
0	The Start Keystroke Intercept function was successful.
1	An incorrect presentation space was specified.
2	An incorrect option was specified.
4	The execution of the function was inhibited because the target presentation space was busy.
9	A system error was encountered. Release is being used.

Notes on Using This Function

1. If a return code of 31 occurs for the Get Key (51) function, either:

   Increase the value of the calling length parameter for this function, or
   Execute the Get Key (51) function more frequently

   An intercepted keystroke occupies three bytes in the buffer. The next intercepted keystroke is placed in the adjacent three bytes. When the Get Key (51) function retrieves a keystroke (first in first out), the 3 bytes that it occupied are made available for another keystroke. By increasing the buffer size or the rate at which keystrokes are retrieved from the buffer, you can eliminate buffer overflow.

   In the PC/3270, another way to eliminate return code 31 is to operate the PC/3270 emulator in the resume mode.

2. If option code *D* is provided, EHLLAPI writes intercepted non-AID keys to the presentation space for which they were originally intended and returns only AID keys to the application.

3. Call the the Stop Keystroke Intercept (53) function before exiting your EHLLAPI application. Otherwise, keystroke interception remains enabled with unpredictable results.

4. For the PC/3270, this function cannot interact with a session that is not declared or with a session that is defined as a logical printer session. See the *Personal Communications Version 4.1 Up and Running for DOS (Full-Function Mode)*, the *Personal Communications Version 4.1 Reference for DOS (Full-Function Mode)*, the *Personal Communications Version 4.1 Up and Running for Windows*, or the *Personal Communications Version 4.1 Reference for Windows*.

Start Playing Macro (110)

PC/3270		*PC400*
DOS	Windows	
–	X	X

This function invokes a macro. The macro will be executed at the connected session.

Prerequisite Calls

Connect Presentation Space (1).

Call Parameters

Function number	Must be 110.
Data string	See the following table.
Length	Must be 10.
PS position	NA.

Byte	Definition
Positions 1–2	Reserved.
Positions 3–10	Null terminated macro name.

Return Parameters

This function returns these valid return codes:.

Return Code	Explanation
0	The Start Playing Macro function was successful.
1	The programs is not connected to a host session.
2	An error was made in specifying parameters.
9	A system error was encountered.

Stop Close Intercept (43)

PC/3270		*PC400*
DOS	Windows	
–	X	X

This function allows the application to turn off the Start Close Intercept (41) function. Once the application has issued the Stop Close Intercept function, subsequent close requests result in a normal stop sent to the logical terminal session.

Prerequisite Calls

Start Close Intercept (41).

Call Parameters

Function number	Must be 43.
Data string	One-character short session ID of the host presentation space.
Data string length	NA.
PS position	NA.

Return Parameters

These return codes are valid:

Return Code	Explanation
0	The Stop Close Intercept function was successful.
1	An incorrect host presentation space was specified.
2	An error was made in specifying parameters.
8	No previous Start Close Intercept function was issued.
9	A system error occurred.
12	The session stopped.

Stop Host Notification (25)

PC/3270		*PC400*
DOS	Windows	
X	X	X

This function disables the capability of the Query Host Update (24) function to determine if the host PS or OIA has been updated. This function also stops host events from affecting the Pause (18) function.

Prerequisite Calls

Start Host Notification (23).

Call Parameters

Function number	Must be 25.
Data string	One-character short name of the target presentation space ID, or a blank or a null to indicate a request against the host-connected presentation space.
Length	NA (1 is implied).
PS position	NA.

Return Parameters

These return codes are valid:

Return Code	Explanation
0	The Stop Host Notification function was successful.
1	An incorrect host presentation space was specified.
8	No previous Start Host Notification (23) function was issued.
9	A system error was encountered.

Stop Keystroke Intercept (53)

PC/3270		*PC400*
DOS	Windows	
X	X	X

This function ends your application program's ability to intercept keystrokes.

Prerequisite Calls

Start Keystroke Intercept (50).

Call Parameters

Function number	Must be 53.
Data string	Short name of the target presentation space (PSID).
Length	NA (1 is implied).
PS position	NA.

Return Parameters

These return codes are valid:

Return Code	Explanation
0	The Stop Keystroke Intercept function was successful.
1	An incorrect presentation space was specified.
8	No prior Start Keystroke Intercept (50) function was called for this presentation space.
9	A system error was encountered.

Storage Manager (17)

PC/3270		PC400
DOS	Windows	
X	–	–

This function is supported only for PC/3270 DOS mode. It allows the PC/3270 EHLLAPI application to perform BASIC language memory functions.

The Storage Manager function allows easy migration for existing applications (typically BASIC interpreter) that use Storage Manager functions. The new BASIC/2 compiler applications can use this function, but are not required to. The other supported languages can also use this function.

During customization, up to 10 KB of memory can be allocated to EHLLAPI in 1-KB increments. See the *Personal Communications Version 4.1 Up and Running for DOS (Full-Function Mode)*, the *Personal Communications Version 4.1 Reference for DOS (Full-Function Mode)*, the *Personal Communications Version 4.1 Up and Running for Windows*, or the *Personal Communications Version 4.1 Reference for Windows*. The block of memory allocated during customization is reserved for EHLLAPI applications when the associated customization batch file is executed. It is available to an application whether or not the Storage Manager function is used. Storage Manager subfunctions can be used to divide this memory into a maximum of 12 areas.

You can make four calls to the storage manager. They are:

Get Storage
Free Storage
Query Free Storage
Free All Storage

These subfunctions are identified to EHLLAPI by passing the subfunction number in the calling PS position parameter.

You can use subfunction 03 Query Storage to determine the largest block of memory available within the block allocated during customization.

You can use subfunction 01 Get Storage to set aside an area of specified size within the block of memory allocated during customization.

You can use subfunction 02 Free Storage to free an area of memory set aside by a specific call to subfunction 01 Get Storage.

You can use subfunction 04 Free All Storage to free up memory in areas set aside by all preceding calls to subfunction 01 Get Storage.

Each of these subfunctions has its own call and return parameters and generates its own set of return codes. Therefore, each subfunction is discussed individually.

Get Storage

This subfunction allocates an area of memory within the storage block reserved during installation. Each time this function is called, a storage block identification number is returned along with a corresponding start address. The storage block identification number is used in calls to subfunction 02 Free Storage to designate a particular area to be freed up. A maximum of 12 areas can be allocated simultaneously.

Prerequisite Calls

None.

Call Parameters

Function number	Must be 17.
Data string	NA.
Length	Size (in bytes) of the requested storage area.
PS position	Specifies subfunction. Pass 01 for Get Storage.

Return Parameters

This subfunction returns a data string, a length, and a return code.

Data string	The storage address expressed as two binary words with the offset address first followed by the segment address.
Length	The storage block ID. Valid range is 01–12, decimal.
Return code	The following codes are valid:

Return Code	Explanation
0	The requested storage was allocated.
1	You have requested more storage than is available.
9	A system error was encountered.

Notes on Using This Function

To use this function, you must preallocate memory to receive the returned data string parameter. The statements required to preallocate this memory vary depending on the language in which your application is written. Refer to “Memory Preallocation” on page 17 for more information.

Free Storage

This subfunction frees a block of storage allocated by subfunction 01 Get Storage.

Prerequisite Calls

None.

Call Parameters

Function number	Must be 17.
Data string	NA.
Length	ID of the storage block to be freed.
PS position	Specifies subfunction. Pass 02 for Free Storage subfunction.

Return Parameters

These return codes are valid:

Return Code	Explanation
0	The storage has been freed.
2	The storage block ID was incorrect.
9	A system error was encountered.

Query Free Storage

This subfunction returns the size of the largest available block of free storage.

Prerequisite Calls

None.

Call Parameters

Function number	Must be 17.
Data string	NA.
Length	NA.
PS position	Specifies subfunction. Pass 03 for Query Free Storage subfunction.

Return Parameters

This subfunction returns a length and a return code.

Length	The size of the largest available block of storage in bytes.
Return code	The following codes are valid:

Return Code	Explanation
0	The Query Free Storage subfunction was successful.
9	A system error was encountered.

Free All Storage

This subfunction frees all allocated storage blocks. The storage blocks are regrouped into a single storage pool that is the same size as the block of memory allocated during installation.

Prerequisite Calls

None.

Call Parameters

Function number	Must be 17.
Data string	NA.
Length	NA.
PS position	Specifies subfunction. Pass 04 for Free All Storage subfunction.

Return Parameters

This subfunction returns a length and a return code.

Length	The size of the original block of memory allocated.
Return code	The following codes are valid:

Return Code	Explanation
0	The Free All Storage subfunction was successful.
9	A system error was encountered.

Wait (4)

PC/3270		PC400
DOS	Windows	
X	X	X

This function checks the status of the host-connected presentation space. If the session is waiting on a host response (indicated by XCLOCK (X []) or XSYSTEM), the Wait function causes EHLLAPI to wait up to one minute to see if the condition clears.

Prerequisite Calls

Connect Presentation Space (1).

Call Parameters

Function number	Must be 4.
Data string	NA.
Length	NA.
PS position	NA.

Return Parameters

These return codes are valid:

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

Notes on Using This Function

1. The Wait function is used to give host requests like those made by the Send Key (3) function the time required to be completed. Using the Set Session Parameters (9) function, you can request the TWAIT, LWAIT or the NWAIT option. See item 13 on page 154.
2. You can use this function to see if the host OIA is inhibited.
3. The Wait function is satisfied by the host unlocking the keyboard. Therefore, a return code of 0 does not necessarily mean that the transaction has been completed. To verify completion of the transaction, you should use the Search Field (30) function or Search Presentation Space (6) function combined with the Wait function to look for expected keyword prompts.

Window Status (104)

PC/3270		PC400
DOS	Windows	
–	X	X

This function allows the application to query or change a windows PS size, location, or visible state.

Prerequisite Calls

Connect Window Services (101).

Call Parameters

Function number	Must be 104.
Data string	See the following table.
Data string length	NA (16 or 20 is implied).
PS position	NA.

The calling data string can contain:

Byte	Definition
Position 1	One-character short session ID of the host presentation space.
Position 2	A request option value, select one of the following values: X'01' for set status X'02' for query for status X'03' for query for extended status

If the request option value is X'01' (set status):

Byte	Definition
Positions 3–4	A 16-bit word containing the status set bits if the request option is 1 (set status). The following codes are valid return values if the request option is set status: X'0001' Change the window size. Not valid with minimize, maximize, restore, or move. X'0002' Move the window. Not valid with minimize, maximize, size, or restore. X'0004' ZORDER window replacement. X'0008' Set the window to visible. X'0010' Set the window to invisible. X'0080' Activate the window. Sets focus to window and places it in foreground unless ZORDER is specified. In this case, the ZORDER placement is used. X'0100' Deactivate the window. Deactivates the window and makes the window the bottom window unless ZORDER is also specified. In this case, the ZORDER placement is used. X'0400' Set the window to minimized. Not valid with maximize, restore, size, or move. X'0800' Set the window to maximized. Not valid with minimize, restore, size, or move. X'1000' Restore the window. Not valid with minimize, maximize, size, or move.
Positions 5–6	A 16-bit word containing the X window position coordinate. Ignored if the move option is not set.
Positions 7–8	A 16-bit word containing the Y window position coordinate. Ignored if the move option is not set.
Positions 9–10	A 16-bit word containing the X window size in device units. Ignored if the size option is not set.

Byte	Definition
Positions 11–12	A 16-bit word containing the Y window size in device units. Ignored if the size option is not set.
Positions 13–16	A 32-bit word containing a window handle for relative window placement. These two words are only for the set option. Ignored if the ZORDER option is not set. Valid values are: X'00000003' Place in front of all sibling windows. X'00000004' Place behind all sibling windows.

If the request option value is X'02' (query for status):

Byte	Definition
Positions 3–4	A 16-bit word containing X'0000' if the request option is 2 (query for status). The following codes are possible return values if the request option is query for status. More than one state is possible. X'0008' The window is visible. X'0010' The window is invisible. X'0080' The window is activated. X'0100' The window is deactivated. X'0400' The window is minimized. X'0800' The window is maximized.
Positions 5–6	A 16-bit word containing the X window position coordinate. Ignored if the move option is not set.
Positions 7–8	A 16-bit word containing the Y window position coordinate. Ignored if the move option is not set.
Positions 9–10	A 16-bit word containing the X window size in device units. Ignored if the size option is not set.
Positions 11–12	A 16-bit word containing the Y window size in device units. Ignored if the size option is not set.
Positions 13–16	A 32-bit word containing a window handle for relative window placement. These two words are only for the set option. Ignored if the ZORDER option is not set. Valid values are: X'00000003' Place in front of all sibling windows. X'00000004' Place behind all sibling windows.

If the request option value is X'03' (query for extended status):

Byte	Definition
Positions 3–4	A 16-bit word containing X'0000' if the request option is 3 (query for extended status). The following codes are possible return values if the request option is query for extended status. More than one state is possible. X'0008' The window is visible. X'0010' The window is invisible. X'0080' The window is activated. X'0100' The window is deactivated. X'0400' The window is minimized. X'0800' The window is maximized.
Positions 5–6	A 16-bit word containing the current font size in the X-dimension. The value is in screen pels.
Positions 7–8	A 16-bit word containing the current font size in the Y-dimension. The value is in screen pels.
Positions 9–10	A 16-bit word containing the X-coordinate offset of the presentation space within the client window. Usually, this value is zero if the automatic sizing option is enabled. The offset is relative to the left edge of the client window.
Positions 11–12	A 16-bit word containing the Y-coordinate offset of the presentation space within the client window. Usually this value is zero if the automatic sizing option is enabled. The offset is relative to the top edge of the client window.

Byte	Definition
Positions 13–14	A 16-bit word containing the row number of the first visible character of the presentation space. This value is usually is ONE, unless the fixed size font option is in effect, and the window has been resized such that some of the presentation space is hidden.
Positions 15–16	A 16-bit word containing the column number of the first visible character of the presentation space.
Positions 17–18	A 16-bit word containing the window handle of the session.
Positions 19–20	Reserved.

Return Parameters

These return codes are valid:

Return Code	Explanation
0	The Window Status function was successful.
1	The presentation space was not valid or not connected.
2	An incorrect option was specified.
9	A system error occurred.
12	The session stopped.

Notes on Using This Function

The logical terminal (LT) windows use character cells. When resizing the LT windows, the LT rounds the number to prevent character cell truncation. The requested size and position might be slightly different from what was requested. Follow the set option with a query option to determine the final window position and size. All x and y coordinate positions and sizes are in pels.

Write Structured Fields (127)

PC/3270		PC400
DOS	Windows	
X	X	–

This function allows an application to write structured field data to the host application. If the call specifies S (for Synchronous), then the application does not receive control until the Write Structured Fields function is completed. If the call specifies A or M (for Asynchronous), then the application receives control immediately after the call. In either case (S, A, or M), the application provides the buffer address in which to data to the host is to be placed.

For a successful asynchronous completion of this function, the following considerations apply:

The return code field in the parameter list might not contain the results of the requested I/O. If the return code is not 0, then the request failed. The application must take the appropriate action based on the return code.

If the return code for this request is 0, the application must use the request ID returned with this function call to issue the Get Request Completion function call to determine the completion results of the function associated with the request ID. The Get Request Completion function call has the following information returned:

- Function request ID
- Address of the data string from the asynchronous request
- Length of the data string
- Return code of the completed function

Prerequisite Calls

Connect for Structured Fields (120)
Allocate Communication Buffer (123).

Call Parameters

Function number	Must be 127.
Data string	See the following table.
Data string length	Length of string (8 or 14).
PS position	NA.

The calling data string can contain:

Byte	Definition
Position 1	A 1-character presentation space short session ID.
Position 2	S, A, or M. S = Synchronous. Control is not returned to the application until the read is satisfied. A = Asynchronous. Control is returned immediately to the application. M = Asynchronous. The asynchronous mode of the notification is requested. (PC/3270 Windows mode only)
Positions 3–4	2-byte destination/origin ID

Byte	Definition
Positions 5–8	4-byte address of the buffer into which the data is to be written. The buffer must be obtained using the Allocate Communications Buffer (123) function.
Positions 9–10	Reserved.
Positions 11–12	When M is specified in position 2, the windows handle of the window that receives the message should be set. (PC/3270 Windows mode only) The message is a return value of: RegisterWindowMessage("PCSHLL") (not equal 0).
Positions 13–14	The data in these position is ignored by EHLLAPI. However, no error is caused if the migrating program has data in these positions. This data is accepted to provide compatibility with migrating applications.

Return Parameters

This function returns a data string and a return code.

Data string — If A (asynchronous) is specified in position 2 and the function is completed successfully, the following data string is returned.

Byte	Definition
Positions 9–10	2-byte function request ID. It is used by the Get Request Completion (125) function to determine the completion of this function call.

If M (asynchronous mode of notification) is specified in position 2 and the function is completed successfully, the following data string is returned.

Byte	Definition
Positions 9–10	2-byte function request ID. It is used by the Get Request Completion (125) function to determine the completion of this function call.
Positions 11–12	Task ID of asynchronous mode of notification.

If the function is completed successfully, an application window receives a message. The message is a return value of RegisterWindowMessage("PCSHLL"). wParam parameter contains the Task ID returned by the function call. HIWORD of lParam parameter contains Return Code 0, which shows the function was successful, and LOWORD of lParam parameter contains function number 127.

These return codes are valid:

Return Code	Explanation
0	The Write Structured Fields function was successful.
1	A specified host presentation space short session ID was incorrect or was not connected.
2	An error was made in specifying parameters.
9	A system error occurred.
11	Resource unavailable (memory unavailable).
34	The message sent inbound to the host was canceled.
35	An outbound transmission from the host was canceled.
36	Request rejected. Lost contact with the host.
37	Failed. Warning: The host is inbound disabled.

Notes on Using This Function

1. Return code 35 is returned when the first Read Structured Fields or Write Structured Fields is requested after an outbound transmission from the host is canceled. Corrective action is the responsibility of the application.
2. Return code 36 requires that the application disconnect from the emulation program and then reconnect to reestablish communications with the host. Corrective action is the responsibility of the application.
3. Return code 37 is returned if the host is inbound disabled. The Write Structured Fields function was successfully requested.
4. The EHLLAPI allows for a maximum of 20 asynchronous requests per application to be outstanding. A return code for unavailable resources is returned if more than 20 asynchronous requests are attempted.

The structured field data format is as follows:

Offset	Length	Contents
0	1 word	X'0000'
2	1 word	m (Message length: The number of bytes data in the message. The number does not include the buffer header prefix, which contains 8 bytes.) This value must be set by application.
4	1 word	X'0000'
6	1 word	X'0000'
8	1 word	Length of the first (or only) structured field message.
10	1 byte	First non–length byte of the structured field message.
		⋮
m+7	1 byte	Last byte in the structured field message.

Bytes 0–7 are the buffer header. These first 8 bytes are used by the emulation program. The user section of the buffer begins with offset 8. Bytes 8 and 9 contain the number of bytes in the first structured field (a structured field message can contain multiple structured fields) including 2 bytes for bytes 8 and 9. Bytes 8–m+7 are used for the structured field message sent to the host.

Write Structured Fields

Chapter 7. PCSAPI Functions

PC/3270 Windows mode and PC400 provide a new API set, called PCSAPI. Although EHLLAPI controls the interaction between a PC application program and host systems after the session is established, PCSAPI can be used to control the PC/3270 or PC400 session itself.

This chapter describes each PCSAPI function in detail. The functions are arranged alphabetically by name.

How to Use PCSAPI

You can write the application program using the PCSAPI in C Language. To develop a PCSAPI application, do as follows:

1. Prepare a source code and add the appropriate PCSAPI calls.
2. Include the header file **PCSAPI.H** in the application program.
3. Compile the source code.
4. Link the resultant **.OBJ** file(s) with the appropriate file.

You also need either to include an import statement in the **.DEF** file, or to link with this call through the PCSAPI import library **PCSAPI.LIB**.

Page Layout Conventions

All PCSAPI function calls are presented in the same format so that you can quickly retrieve the information you need. The format for each function name is:

Function type
Parameter type and description
Return code

Function Type

"Function Type" uses this format:

TYPE FunctionName(*TYPE Parameter1, ...***)**

Parameter Type and Description

"Parameter Type and Description" describes the type and description of the parameters to be specified in the PCSAPI function call.

Return Code

"Return Code" lists the codes that must be received by your program after a call to the discussed PCSAPI function.

© Copyright IBM Corp. 1996

pcsQueryEmulatorStatus

PC/3270		*PC400*
DOS	Windows	
–	X	X

The **pcsQueryEmulatorStatus** function returns a status of the host session specified by the short session ID,

Function Type

WORD FAR PASCAL pcsQueryEmulatorStatus(*char cShortSessionID***)**

Parameter Type and Description

char cShortSessionID PS short session ID.

Return Code

The return code value should be processed bit-significantly. That is, one of the following values, or an ORed value.

Return Code	Value	Meaning
SESSION_STARTED	0x0001	Specified session has started. When this bit is off, the specified session has not started or an incorrect session ID was specified.
SESSION_ONLINE	0x0002	Specified session is online. When this bit is off, the specified session is offline.
SESSION_API_ENABLED	0x0004	API (EHLLAPI, DDE) is enabled on the specified session. If this bit is off, API is disabled on this session.

pcsQueryWorkstationProfile

PC/3270		*PC400*
DOS	Windows	
–	X	X

The **pcsQueryWorkstationProfile** function returns the name of the workstation profile that invoked the host session. To specify the host session, the short session ID must be used. The workstation profile name is copied to the work buffer prepared by the user.

Function Type

BOOL FAR PASCAL pcsQueryWorkstationProfile(*char cShortSessionID, LPSTR lpBuffer***)**

Parameter Type and Description

char cShortSessionID PS short session ID.

LPSTR lpBuffer Work buffer to copy a null-terminated Workstation Profile name.

Return Code

Return Code	Meaning
TRUE	Function ended successfully.
FALSE	Session has not started or an incorrect session ID was specified.

pcsStartSession

PC/3270		PC400
DOS	Windows	
–	X	X

The **pcsStartSession** function starts a host session by using a specified workstation profile (**.WS**). A short session ID also can be specified.

Function Type

WORD FAR PASCAL pcsStartSession(*LPSTR lpProfile, char cShortSessionID, WORD fuCmdShow***)**

Parameter Type and Description

LPSTR lpProfile (Path and) name of profile to load.

char cShortSessionID PS short session ID. A blank space or NULL indicates the next available session ID.

WORD fuCmdShow Specifies how the window is to be displayed. This parameter can be one of the following values:

fuCmdShow	Value	Meaning
SW_HIDE	0	Hides the window and passes activation to another window.
SW_MINIMIE	6	Minimizes the specified window in the system's list.
SW_RESTORE	9	Activates and displays a window. If the window is minimized or maximized, Windows restores it to its original size and position (same as SW_SHOWNORMAL).
SW_SHOW	5	Activates a window and displays it in its current size and position.
SW_SHOWMAXIMIZED	3	Activates a window and displays it as a maximized window.
SW_SHOWMINIMIZED	2	Activates a window and displays it as an icon.
SW_SHOWMINNOACTIVE	7	Displays a window as an icon. The currently active window remains active.
SW_SHOWNA	8	Displays a window in its current state. The currently active window remains active.
SW_SHOWNOACTIVATE	4	Displays a window in its most recent size and position. The window that is currently active remains active.
SW_SHOWNORMAL	1	Activates and displays a window. If the window is minimized or maximized, Windows restores it to its original size and position (same as SW_RESTORE).

Return Code

Return Code	Value	Meaning
SS_SUCCESSFUL	0	Function ended successfully.
SS_INVALID_ID	1	An incorrect session ID was specified.
SS_USED_ID	2	Specified short session ID is already used.
SS_INVALID_PROFILE	3	An error was made in specifying workstation profile.
SS_SYSTEM_ERROR	9	A system error occurred.

pcsStopSession

PC/3270		PC400
DOS	Windows	
–	X	X

The **pcsStopSession** function stops a host session specified by the session short ID.

Function Type

BOOL FAR PASCAL pcsStopSession(*char cShortSessionID, WORD fuSaveProfile***)**

Parameter Type and Description

char cShortSessionID PS short session ID.

WORD fuSaveProfile This parameter can be one of the following values.

fuSaveProfile	Value	Meaning
SS_SAVE_AS_PROFILE	0	Save a profile as specified in the current profile.
SS_SAVE_ON_EXIT	1	Save a profile on exit.
SS_NOSAVE_ON_EXIT	2	Do not save a profile on exit.

Return Code

Return Code	Meaning
TRUE	Function ended successfully.
FALSE	Session has not started or an incorrect session ID was specified.

pcsStopSession

Chapter 8. DDE Functions

PC/3270 Windows mode and PC400 provide a dynamic data exchange (DDE) interface that allows applications to exchange data. The exchange of data between two Windows applications can be thought of as a conversation between a client and a server. The *client* initiates DDE conversations. The *server* in turn responds to the client. PC/3270 and PC400 is a DDE server for the open sessions that PC/3270 and PC400 are managing. For more information about DDE, refer to *Microsoft Windows Software Development Kit Guide to Programming*.

Note: If you use DDE functions with Visual Basic, see Chapter 9, "Using DDE Functions with DDE Client Application (Visual Basic)" on page 281.

PC/3270 and PC400 DDE Data Items

Microsoft Windows DDE uses a three-level naming scheme to identify data items: application, topic, and item. Table 14 on page 187 describes these levels.

Table 14. Naming Scheme for Data Items

Level	Description	Example
Application	A Windows task or a particular task of an application. In this book, the application is PC/3270 or PC400.	IBM3270
Topic	A specific part of an application.	SessionA
Item	A data object that can be passed in a data exchange. An item is an application-defined data item that conforms to one of the Windows clipboard formats or to a private, application-defined, clipboard format. For more information regarding Windows clipboard formats, refer to *Microsoft Windows Software Development Kit Guide to Programming*.	PS (presentation space)

PC/3270 and PC400 support one application name as Windows DDE server.

Table 15. Applications for PC/3270 and PC400

Application
IBM3270
IBM5250

© Copyright IBM Corp. 1996

The following table lists the topics you can use.

Table 16. Topics for PC/3270 and PC400

Topics
System SessionA, SessionB, ..., SessionZ LUA_xxxx, LUB_xxxx, ..., LUZ_xxxx

In DDE, *atoms* identify application names, topic names, and data items. Atoms represent a character string that is reduced to a unique integer value. The character string is added to an atom table, which can be referred to for the value of the string associated with an atom. Atoms are created with the GlobalAddAtom function call. Refer to *Microsoft Windows Software Development Kit Guide to Programming* for more information about how to create and use atoms.

Using System Topic Data Items

Applications that provide a DDE interface should also provide a special topic SYSTEM. This topic provides a context for items of information that might be of general interest to an application. The SYSTEM topic for PC/3270 or PC400 contains these associated data items:

Item	Function
Formats	Returns the list of clipboard formats (numbers) that PC/3270 or PC400 is capable of rendering.
Status	Returns information about the status of each PC/3270 or PC400 session.
SysCon	Returns the level of PC/3270 or PC400 support and other system related values.
SysItems	Returns the list of data items that are available when connected to the PC/3270 or PC400 system topic.
Topics	Returns the list of PC/3270 or PC400 topics that are available.

Using Session Topic Data Items

For each Session topic, the following data items are supported:

Item	Function
CONV	Converts ASCII to EBCDIC or EBCDIC to ASCII.
CLOSE	Retrieves the window close requests.
EPS	Retrieves the session presentation space with additional data.
EPSCOND	Retrieves the presentation space service condition.
FIELD	Retrieves the field in the presentation space of the session.
KEYS	Retrieves the keystrokes.
MOUSE	Retrieves the mouse input.
OIA	Retrieves the operator information area status line.
PS	Retrieves the session presentation space.
PSCOND	Retrieves the session advise condition.
SSTAT	Retrieves the session status.
STRING	Retrieves the ASCII string data.
TRIMRECT	Retrieves the session presentation space within the current trim rectangle.

Using LU Topic Data Items (PC/3270 Only)

For each LU topic, the following data items are supported:

Item	Function
SF	Retrieves the destination/origin structured field data.
SFCOND	Retrieves the query reply data.

DDE Functions

The following table lists the DDE functions that are available for use with PC/3270 or PC400 and the page in this section, where each function is more fully documented.

Function	PC/3270 Windows	PC400	Page
Code Conversion	X	X	192
Find Field	X	X	193
Get Keystrokes	X	X	195
Get Mouse Input	X	X	197
Get Number of Close Requests	X	X	200
Get Operator Information Area	X	X	201
Get Partial Presentation Space	X	X	202
Get Presentation Space	X	X	205
Get Session Status	X	X	208
Get System Configuration	X	X	210
Get System Formats	X	X	212
Get System Status	X	X	213
Get System SysItems	X	X	214
Get System Topics	X	X	215
Get Trim Rectangle	X	X	216
Initiate Session Conversation	X	X	217
Initiate Structured Field Conversation	X	–	218
Initiate System Conversation	X	X	219
Put Data to Presentation Space	X	X	220
Search for String	X	X	221
Send Keystrokes	X	X	222
Session Execute Macro	X	X	223
Set Cursor Position	X	X	230
Set Mouse Intercept Condition	X	X	231
Set Presentation Space Service Condition	X	X	234
Set Session Advise Condition	X	X	236
Set Structured Field Service Condition	X	–	238
Start Close Intercept	X	X	239
Start Keystroke Intercept	X	X	241
Start Mouse Input Intercept	X	X	242
Start Read SF	X	–	246
Start Session Advise	X	X	249
Stop Close Intercept	X	X	251
Stop Keystroke Intercept	X	X	252
Stop Mouse Input Intercept	X	X	253
Stop Read SF	X	–	254
Stop Session Advise	X	X	255
Terminate Session Conversation	X	X	256
Terminate Structured Field Conversation	X	–	257
Terminate System Conversation	X	X	258
Write SF	X	–	259

Refer to “Summary of DDE Functions” on page 260 for a summary of the DDE functions.

Naming Conventions for Parameters

Most DDE parameter names and local variables. These variables have a prefix that indicates the general type of the parameter, followed by one or more words that describe the content of the parameter. Prefixes presented in this book are:

a	Atom
c	Character (a 1-byte value)
f	Bit flags packed into a 16-bit integer
h	16-bit handle
p	Short (16-bit) pointer
lp	Long (32-bit) pointer
w	Short (16-bit) unsigned integer
u	Unsigned integer
sz	Null-terminated character string

Code Conversion

PC/3270		PC400
DOS	Windows	
–	X	X

The Code Conversion function allow a client application to convert ASCII to EBCDIC or EBCDIC to ASCII. The client sends the following message for the field information.

```
PostMessage( hServerWnd,
             WM_DDE_POKE,
             hClientWnd,
              MAKELONG(hData, aCONV) );
```

where:

hData Identifies a handle to a Windows global memory object. The global memory object contains the following structure:

```
typedef struct tagCONVDATA
{
 unsigned    unused:13;          //    unused
 unsigned    fRelease:1;         //
 unsigned    freserved:2;        //    Reserved
 int         cfFormat;           // CF_DSPTEXT
 HANDLE      hSourceBuffer;      // handle of global memory object
                                 // to store the data to convert
 HANDLE      hTargetBuffer;      // handle of global memory object
                                 // to store the converted data
 BYTE        ConvType;           // Conversion type (refer to NOTE below)
 unsigned    uSourceLength;      // string length (source)
 unsigned    uTargetLength;      // buffer size of the converted data
 unsigned    uStringLength;      // string length of the converted data (returned)
}
```

The following conversion types indicated by ConvType are:

X'01'—ASCII to EBCDIC
X'02'—EBCDIC to ASCII

aCONV Identifies code conversion item.

PC/3270 or PC400 Response

PC/3270 or PC400 converts ASCII to EBCDIC or EBCDIC to ASCII. If PC/3270 or PC400 cannot convert the data, a DDE ACK message is returned with an error code in the low-order byte of the wStatus word:

Return Code	Explanation
0	Normal end. The latest converted code is SBCS.
2	The specified conversion type is not valid.
9	A system error occurred.
10	The destination buffer exceeded.
11	Internal translation error occurred.

Find Field

PC/3270		PC400
DOS	Windows	
–	X	X

The find field function returns to the client the information of the field specified by the set presentation space service condition function.

Note: The client must set the PS position by using the set presentation service condition function before using this function.

The client sends this following message to receive the field information.

```
PostMessage( hServerWnd,
             WM_DDE_REQUEST,
             hClientWnd,
              MAKELONG(cfFormat, aFIELD) );
```

where:

cfFormat Identifies the format for the field information.
Valid value must be CF_DSPTEXT.

aFIELD Identifies field data item.

The Find Field function supports a new format like Visual Basic. Using the new format, the Find Field function can find a field with specifying its type. The new format is:

FIELD (pos, type)

pos Position where PC/3270 or PC400 starts to search a target field.
type Target field type. The field type are:

Type	Meaning
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.

Note: The symbol represents a required blank.

PC/3270 or PC400 Response

PC/3270 or PC400 returns the following information of the field in a DDE data message,

Start PS position
Length
Attribute value

WM_DDE_DATA(hData, aFIELD)

or responds with an ACK message containing status information.

WM_DDE_ACK(wStatus, aFIELD)

If PC/3270 or PC400 cannot return the field information, the following status codes will be returned in the low order byte of the wStatus word:

Return Code	Explanation
1	PS position is not valid.
2	PS is unformatted.
6	The specified format is not valid.
9	A system error occurred.

Structure of the Field Information

PC/3270 or PC400 returns the field information in the following structure:

```
typedef struct tagFINDFIELD
{                                   //
 unsigned unused:12;                //     unused
 unsigned fResponse:1;              // TRUE = DDE_REQUEST response
 unsigned fRelease:1;               // TRUE = Client frees this data
 unsigned reserved:1;               //     reserved
 unsigned fAckReq:1;                // TRUE = Client returns DDE_ACK
 int          cfFormat;             // Format of Field data CF_DSPTEXT
 unsigned char cAttribute;    // Attribute character
 unsigned uFieldStart;              // Field start offset
 unsigned uFieldLength;             // Field Length;
}    FINDFIELD, far  lpFINDFIELD;
```

Get Keystrokes

PC/3270		*PC400*
DOS	Windows	
–	X	X

The Get Keystrokes function returns to the client the keystrokes that are intercepted by the Start Keystroke Intercept function. The client sends the following message to receive the keystroke information.

```
PostMessage( hServerWnd,
             WM_DDE_REQUEST,
             hClientWnd,
              MAKELONG(cfFormat, aKEYS) );
```

where:

cfFormat Identifies the format for the keystroke information. Valid value must be CF_DSPTEXT.

aKEYS Identifies keystroke data item.

PC/3270 or PC400 Response

PC/3270 or PC400 either returns the keystrokes in a DDE data message, or responds with one of these ACK messages containing status information:

WM_DDE_DATA(hData, aKEYS)
WM_DDE_ACK(wStatus, aKEYS)

If PC/3270 or PC400 cannot return the keystroke information, the following status codes will be returned in the low-order byte of the wStatus word:

Return Code	Explanation
2	No keystroke was intercepted.
6	The specified format is not valid.
9	A system error occurred.

Structure of the Keystroke Information

PC/3270 or PC400 returns the keystroke information in the following structure:

```
typedef struct tagKEYSTROKE
{                                   //
 unsigned unused:12;                //     unused
 unsigned fResponse:1;              // TRUE = DDE_REQUEST response
 unsigned fRelease:1;               // TRUE = Client frees this data
 unsigned reserved:1;               //     reserved
 unsigned fAckReq:1;                // TRUE = Client returns DDE_ACK
 int            cfFormat;           // Format of keystroke data CF_DSPTEXT
 unsigned uTextType;                // Type of keystrokes
 unsigned char szKeyData[1];// Keystrokes
}    KEYSTROKE, far  lpKEYSTROKE;
```

The format for the keystrokes parameters is the same as for the Session Execute Macro function SENDKEY command.

The following key text types are supported:

```
WC_CHARACTER      // Pure text, no command
WC_TOKEN          1 // including commands
```

Get Mouse Input

PC/3270		*PC400*
DOS	Windows	
–	X	X

The Get Mouse Input function returns the latest mouse input that are intercepted by Start Mouse Input Intercept function to the client.

Note: The client must call the Start Mouse Input Intercept function before using this function.

The client sends the following command to receive the mouse input information.

```
PostMessage( hServerWnd,
             WM_DDE_REQUEST,
             hClientWnd,
              MAKELONG(cfFormat, aMOUSE) );
```

where:

cfFormat Identifies the format for the presentation space. Valid values are CF_TEXT or CF_DSPTEXT. The structure of the mouse input data, in these two formats, is shown below.

aMOUSE Identifies the mouse input as the item.

PC/3270 or PC400 Response

PC/3270 or PC400 either returns the mouse input data in a DDE data message, or responds with one of these ACK messages:

WM_DDE_DATA(hData, aMOUSE)
WM_DDE_ACK(wStatus, aMOUSE)

If PC/3270 or PC400 cannot return the mouse input information, the following status codes are returned in the low-order byte of the wStatus word:

Return Code	Explanation
2	No mouse input information was intercepted.
6	The specified format is not valid.
9	A system error occurred.

Structure of the Mouse Input Information

If the format is CF_TEXT, PC/3270 or PC400 returns the mouse input information in the following format:

```
typedef struct tagMOUSE_CF_TEXT
{
 unsigned          unused:12,          //     Unused
  unsigned         fRespons:1,        // TRUE = DDE_REQUEST response
  unsigned         fRelease:1,        // TRUE = Client frees this data
  unsigned         reserved:1,        //       Reserved
 unsigned          fAckReq:1;           // TRUE = Client returns DDE_ACK
 int               cfFormat;            // CF_TEXT
 unsigned char PSPos[4];              // PS position
 unsigned char Tab1[1];               // TAB character
 unsigned char PSRowPos[4];           // PS row position
 unsigned char Tab2[1];               // TAB character
 unsigned char PSColPos[4];           // PS columns position
 unsigned char Tab3[1];               // TAB character
 unsigned char PSSize[4];             // Size of the PS
 unsigned char Tab4[1];               // TAB character
 unsigned char PSRows[4];             // PS number of rows
 unsigned char Tab5[1];               // TAB character
 unsigned char PSCols[4];             // PS number of columns
 unsigned char Tab6[1];               // TAB character
 unsigned char ButtonType[1];         // Pressed button type
 unsigned char Tab7[1];               // TAB character
 unsigned char ClickType[1];          // Click type
 unsigned char Tab8[1];               // TAB character
 unsigned char ClickString[1];    // Retrieved string
} MOUSE_CF_TEXT, FAR  lpMOUSE_CF_TEXT;
```

The following table shows the values in the parameters:

Parameter Name	Meaning	Value
PSPos	PS offset of the position where mouse was clicked	0 ... (PSSize – 1)
PSRowPos	Row number of the position where mouse was clicked	0 ... (PSRows – 1)
PSColPos	Column number of the position where mouse was clicked	0 ... (PSCols – 1)
PSSize	Size of the presentation space	
PSRows	Number of rows of presentation space	
PSCols	Number of columns of presentation space	
ButtonType	Type of the clicked mouse button	**L** Left button **M** Middle button **R** Right button
ClickType	Type of clicking	**S** Single click **D** Double click
ClickString	Retrieved string to which the mouse pointed	A character string terminated with a '\0'
Tab1–8	A tab character for delimiter	'\t'

If the format is CF_DSPTEXT, PC/3270 or PC400 returns the mouse input information in the following format:

```
typedef struct tagMOUSE_CF_DSPTEXT
{
 unsigned        unused:12,          //     Unused
 unsigned        fRespons:1,         // TRUE = DDE_REQUEST response
 unsigned        fRelease:1,         // TRUE = client frees the storage
 unsigned        reserved:1,         //     Reserved
 unsigned        fAckReq:1;          // TRUE = client returns DDE_ACK
 int             cfFormat;           // CF_DSPTEXT
 unsigned        uPSPos;             // PS position
 unsigned        uPSRowPos;          // PS row position
 unsigned        uPSColPos;          // PS column position
 unsigned        uPSSize;            // Size of the presentation space
 unsigned        uPSRows;            // PS number of rows
 unsigned        uPSCols;            // PS number of columns
 unsigned        uButtonType;        // Pressed button type
 unsigned        uClickType;         // Click type
 unsigned char szClickString[1];   // Retrieved string
} MOUSE_CF_DSPTEXT, FAR  lpMOUSE_CF_DSPTEXT;
```

The following table shows the values in the parameters:

Parameter Name	Meaning	Value
uPSPos	PS offset of the position where mouse was clicked	0 ... (uPSSize – 1)
uPSRowPos	Row number of the position where mouse was clicked	0 ... (uPSRows – 1)
uPSColPos	Column number of the position where mouse was clicked	0 ... (uPSCols – 1)
uPSSize	Size of the presentation space	
uPSRows	Number of rows of the presentation space	
uPSCols	Number of columns of the presentation space	
uButtonType	Type of the clicked mouse button	**0x0001** Left button **0x0002** Middle button **0x0003** Right button
uClickType	Type of clicking	**0x0001** Single click **0x0002** Double click
szClickString	Retrieved string that the mouse pointed to	A character string terminated with a '\0'

Get Number of Close Requests

PC/3270		*PC400*
DOS	Windows	
–	X	X

The Get Number of Close Requests function returns to the client the number of the close requests that are intercepted by the Start Close Intercept function. The client sends the following message to receive the number of the close requests.

```
PostMessage( hServerWnd,
             WM_DDE_REQUEST,
             hClientWnd,
              MAKELONG(cfFormat, aCLOSE) );
```

where:

cfFormat Identifies the format for the close intercept information. Valid value must be CF_DSPTEXT.

aCLOSE Identifies close intercept data item.

PC/3270 or PC400 Response

PC/3270 or PC400 either returns the number of the close requests in a DDE data message, or responds with one of these ACK messages:

WM_DDE_DATA(hData, aCLOSE)
WM_DDE_ACK(wStatus, aCLOSE)

If PC/3270 or PC400 cannot return the close intercept information, the following status codes will be returned in the low order byte of the wStatus word:

Return Code	Explanation
6	The specified format is not valid.
9	A system error occurred.

Structure of the Number of the Close Requests Information

PC/3270 or PC400 returns the close intercept information in the following structure:

```
typedef struct tagCLOSEREQ
{                                       //
 unsigned unused:12;                 //     unused
 unsigned fResponse:1;              // TRUE = DDE_REQUEST response
 unsigned fRelease:1;               // TRUE = Client frees this data
 unsigned reserved:1;               //     reserved
 unsigned fAckReq:1;                  // TRUE = Client returns DDE_ACK
 int          cfFormat;               // Format of close intercept data CF_DSPTEXT
 unsigned uCloseReqCount;      // Number of the close requests.
}    CLOSEREQ, far  lpCLOSEREQ;
```

Get Operator Information Area

PC/3270		PC400
DOS	Windows	
–	X	X

The Get Operator Information Area (OIA) function returns a copy of the OIA to the client. The client sends the following message to request the OIA.

```
PostMessage( hServerWnd,
             WM_DDE_REQUEST,
             hClientWnd,
              MAKELONG(cfFormat, aOIA) );
```

where:

cfFormat Identifies the format for the OIA. For the OIA this format must be CF_DSPTEXT.

aOIA Identifies the operator information area as the item.

PC/3270 or PC400 Response

PC/3270 or PC400 either returns the OIA in a DDE data message, or responds with one of these ACK messages:

WM_DDE_DATA(hData, aOIA)
WM_DDE_ACK(wStatus, aOIA)

If PC/3270 or PC400 cannot return the OIA, the following status codes will be returned in the low order byte of the wStatus word:

Return Code	Explanation
6	The specified format is not valid.
9	A system error occurred.

Structure of the Operator Information Area

PC/3270 or PC400 returns the operator information area in the following structure:

```
typedef struct tagOIADATA
{                                   //
 unsigned unused:12;              //    unused
 unsigned fResponse:1;           // TRUE = DDE_REQUEST response
 unsigned fRelease:1;            // TRUE = Client frees this data
 unsigned reserved:1;            //    reserved
 unsigned fAckReq:1;               // TRUE = Client returns DDE_ACK
 int    cfFormat;                  // Format of OIA data CF_DSPTEXT
 unsigned char OIA[ ];           //
}    OIADATA, far  lpOIADATA;
```

Get Partial Presentation Space

PC/3270		PC400
DOS	Windows	
–	X	X

The Get Partial Presentation Space function returns all or part of the session presentation space to the client.

Note: The client must set the start PS position and the length (or set the EOF flag) by using the Set Presentation Space Service Condition function before using this function.

The client sends the following command to get the presentation space.

```
PostMessage( hServerWnd,
             WM_DDE_REQUEST,
             hClientWnd,
              MAKELONG(cfFormat, aEPS) );
```

where:

cfFormat Identifies the format for the presentation space. Valid values are CF_TEXT or CF_DSPTEXT. The structure of the presentation space, in these two formats, is shown below.

aEPS Identifies presentation space atom as the item.

PC/3270 or PC400 Response

PC/3270 or PC400 either returns the presentation space data, or responds with one of these ACK messages containing an error code in the low order byte of the wStatus word:

WM_DDE_DATA(hData, aEPS)
WM_DDE_ACK(wStatus, aEPS)

If PC/3270 or PC400 cannot return the presentation space, the following status codes will be returned in the low order byte of the wStatus word:

Return Code	Explanation
1	No prior Set Presentation Space Service Condition function was called, or an incorrect parameter was set.
6	The specified format is not valid.
9	A system error occurred.

Structure of the Presentation Space

PC/3270 or PC400 returns the part of the presentation space in the format specified in the Get Partial Presentation Space request.

If the format is CF_DSPTEXT, PC/3270 or PC400 returns the presentation space in the following format:

```
typedef struct tagEPS_CF_DSPTEXT
{
 unsigned Unused:12,                  // Unused
 unsigned fResponse:1,                // TRUE = DDE_REQUEST response
 unsigned fRelease:1,                 // TRUE = client frees the storage
 unsigned reserved:1,                 //      Reserved
 unsigned fAckReq:1,                  // TRUE = DDE_ACK requested
 int          cfFormat;               // Format data is rendered in
 unsigned uPSPosition;                // Start PS position
 unsigned uPSLength;                  // Length of the part of the PS
 unsigned uPSRows;                    // PS number of rows
 unsigned uPSCols;                    // PS number of columns
 unsigned uPSOffset;                  // Offset to the presentation space
 unsigned uFieldCount;                // Number of fields
 unsigned uFieldOffset;               // Offset to the field array
 unsigned char PSData[1];             // PS and Field list Array
} EPS_CF_DSPTEXT, FAR  lpEPS_CF_DSPTEXT;

typedef struct tagPSFIELDS
{
 unsigned char cAttribute;          // Attribute Character
 unsigned uFieldStart;              // Field start offset
 unsigned uFieldLength;             // Field Length
} PSFIELDS, FAR  lpPSFIELDS;
```

Note: The following examples show how to obtain long pointers to the PS and the PSFIELDS array.

```
  lpps = (lp_EPS_CF_DSPTEXT) lpEPS_CF_DSPTEXT->PSData
       +  lpEPS_CF_DSPTEXT->uPSOffset;
  lppsfields = (lpPSFIELDS) lpEPS_CF_DSPTEXT->PSData
                 +  lpEPS_CF_DSPTEXT->uFieldOffset;
```

If the format is CF_TEXT, PC/3270 or PC400 returns the part of the presentation space in the following format:

```
typedef struct tagEPS_CF_TEXT
{
 unsigned          Unused:12;          //     Unused
 unsigned          fResponse:1;       // TRUE = DDE_REQUEST response
  unsigned         fRelease:1;        // TRUE = Client frees this data
  unsigned         reserved:1;        //       Reserved
 unsigned          fAckReq:1;          // TRUE = Client returns DDE_ACK
 int               cfFormat;           // Format of the data
 unsigned char PSPOSITION[4];    // Start PS position
 unsigned char Tab1[1];             // Tab character
 unsigned char PSLENGTH[4];       // Length of the part of the PS
 unsigned char Tab2[1];             // Tab character
 unsigned char PSROWS[4];          // Number of rows in the Partial PS
 unsigned char Tab3[1];             // Tab character
 unsigned char PSCOLS[4];         // Number of columns in the PS
 unsigned char Tab4[1];             // Tab character
 unsigned char PS[1];               // PS
} EPS_CF_TEXT, FAR  lpEPS_CF_TEXT;
```

Following the PS in the buffer is the following additional structure of fields that compose the field list.

```
typedef struct tagFL_CF_TEXT
{
 unsigned char Tab5[1];            // Tab character
 unsigned char PSFldCount[4]; // Number of fields in the PS
 unsigned char Tab6[1];            // Tab character
  PS_FIELD         Field[1];        // Field List Array
} FL_CF_TEXT, FAR  lpFL_CF_TEXT;

typedef struct tagPS_FIELD
{
 unsigned char FieldStart[4];
 unsigned char TabF1[1];
 unsigned char FieldLength[4];
 unsigned char TabF2[1];
 unsigned char Attribute;
 unsigned char TabF3[1[;
} PS_FIELD, FAR  lpPS_FIELD;
```

Note: The following examples show how to obtain long pointers to the PS and the PS_FIELD array.

```
  lpps = lpEPS_CF_TEXT->PS;
  lpps_field = (lpPS_FIELD) lpEPS_CF_TEXT->PS
                + atoi(lpEPS_CF_TEXT->PSLENGTH)
                 + ((atoi(lpEPS_CF_TEXT->PSROWS) - 1)   2) // CR/LF
                 + 1 + 1 + 4 + 1;      // Tabs + size of field count
```

Get Presentation Space

PC/3270		*PC400*
DOS	Windows	
–	X	X

The Get Presentation Space function returns the session presentation space to the client. The client sends the following command to get the presentation space.

```
PostMessage( hServerWnd,
             WM_DDE_REQUEST,
             hClientWnd,
              MAKELONG(cfFormat, aPS) );
```

where:

cfFormat Identifies the format for the presentation space. Valid values are CF_TEXT or CF_DSPTEXT. The structure of the presentation space, in these two formats, is shown below.

aPS Identifies presentation space atom as the item.

PC/3270 or PC400 Response

PC/3270 or PC400 either returns the presentation space and a list of the fields that comprise the presentation space, or responds with one of these ACK messages containing an error code in the low order byte of the wStatus word:

WM_DDE_DATA(hData, aPS)
WM_DDE_ACK(wStatus, aPS)

If PC/3270 or PC400 cannot return the presentation space, the following status codes will be returned in the low order byte of the wStatus word:

Return Code	Explanation
4	The specified format is not valid.
9	A system error occurred.

Structure of the Presentation Space

PC/3270 or PC400 returns the presentation space in the format specified in the Get Presentation Space request.

If the format is CF_DSPTEXT, PC/3270 or PC400 returns the presentation space in the following format:

```
typedef struct tagPS_CF_DSPTEXT
{
 unsigned Unused:12,                // Unused
 unsigned fResponse:1,              // TRUE = DDE_REQUEST response
 unsigned fRelease:1,               // TRUE = client frees the storage
 unsigned reserved:1,               //      Reserved
 unsigned fAckReq:1,                // TRUE = DDE_ACK requested
 int          cfFormat;             // Format data is rendered in
 unsigned uPSSize;                  // Size of the presentation space
 unsigned uPSRows;                  // PS number of rows
 unsigned uPSCols;                  // PS number of columns
 unsigned uPSOffset;                // Offset to the presentation space
 unsigned uFieldCount;              // Number of fields
 unsigned uFieldOffset;             // Offset to the field array
 unsigned char PSData[1];           // PS and Field list Array
} PS_CF_DSPTEXT, FAR  lpPS_CF_DSPTEXT;

typedef struct tagPSFIELDS
{
 unsigned char cAttribute;          // Attribute Character
 unsigned uFieldStart;              // Field start offset
 unsigned uFieldLength;             // Field Length
} PSFIELDS, FAR  lpPSFIELDS;
```

Note: The following examples show how to obtain long pointers to the PS and the PSFIELDS array.

```
lpps = (lp_PS_CF_DSPTEXT) lpPS_CF_DSPTEXT->PSData
       +  lpPS_CF_DSPTEXT->uPSOffset;
lppsfields = (lpPSFIELDS) lpPS_CF_DSPTEXT->PSData
             +  lpPS_CF_DSPTEXT->uFieldOffset;
```

If the format is CF_TEXT, PC/3270 or PC400 returns the presentation space in the following format:

```
typedef struct tagPS_CF_TEXT
{
 unsigned          Unused:12;          //      Unused
 unsigned          fResponse:1;        // TRUE = DDE_REQUEST response
  unsigned         fRelease:1;         //  TRUE  =  Client  frees  this  data
  unsigned         reserved:1;         //       Reserved
 unsigned          fAckReq:1;          // TRUE = Client returns DDE_ACK
 int               cfFormat;           // Format of the data
 unsigned char PSSIZE[4];              // Size of the PS
 unsigned char Tab1[1];                // Tab character
 unsigned char PSROWS[4];              // Number of rows in the PS
 unsigned char Tab2[1];                // Tab character
 unsigned char PSCOLS[4];              // Number of Cols in the PS
 unsigned char Tab3[1];                // Tab character
 unsigned char PS[1];                  // PS
} PS_CF_TEXT, FAR  lpPS_CF_TEXT;
```

Following the PS in the buffer is the following additional structure of fields that compose the field list.

```
typedef struct tagFL_CF_TEXT
{
 unsigned char Tab4[1];              // Tab character
 unsigned char PSFldCount[4]; // Number of fields in the PS
 unsigned char Tab5[1];              // Tab character
  PS_FIELD          Field[1];          //  Field  List  Array
} FL_CF_TEXT, FAR  lpFL_CF_TEXT;

typedef struct tagPS_FIELD
{
 unsigned char FieldStart[4];
 unsigned char TabF1[1];
 unsigned char FieldLength[4];
 unsigned char TabF2[1];
 unsigned char Attribute;
 unsigned char TabF3[1];
} PS_FIELD, FAR  lpPS_FIELD;
```

Note: The following example shows how to obtain long pointers to the PS and the PS_FIELD array.

```
   lpps = lpPS_CF_TEXT->PS;
   lpps_field = (lpPS_FIELD) lpPS_CF_TEXT->PS
                    +  atoi(lpPS_CF_TEXT->PSSIZE)
                     + ((atoi(lpPS_CF_TEXT->PSROWS) - 1)   2) // CR/LF
                     + 1 + 1 + 4 + 1;      // Tabs + size of field count
```

Get Session Status

PC/3270		PC400
DOS	Windows	
–	X	X

The Get Session Status function returns the status of the connected session. The client sends the following message to request session status:

```
PostMessage( hServerWnd,
             WM_DDE_REQUEST,
             hClientWnd,
              MAKELONG(cfFormat, aSSTAT) );
```

where:

cfFormat Identifies the DDE format for the status information. The value used is CF_TEXT.

aSSTAT Identifies session status as the data item requested.

PC/3270 or PC400 Response

PC/3270 or PC400 either returns the session status in a DDE data message, or responds with one of these ACK messages containing status information:

WM_DDE_DATA(hData, aSSTAT)
WM_DDE_ACK(wStatus, aSSTAT)

If PC/3270 or PC400 cannot return the session status, the following status codes will be returned in the low order byte of the wStatus word:

Return Code	Explanation
6	The specified format is not valid.
9	A system error occurred.

Format of Status Information

PC/3270 or PC400 returns the session status as text in CF_TEXT format. The following fields are returned with the following possible values:

Fields	Returned values	Description
Status	Closed, Invisible, Maximized, Minimized, Normal	The window is in one of these states.
Usage	DDE, User	The session is connected in either a DDE session or a user session.
ScreenX	NN	Defines the horizontal size of the screen.
ScreenY	NN	Defines the vertical size of the screen.
CursorX	NN	Defines the horizontal position of the cursor. (0 ... ScreenX – 1)

Fields	Returned values	Description
CursorY	NN	Defines the vertical position of the cursor. (0 ... ScreenY – 1)
TrimRect Status	Closed, Moved, Sized	The current status of the trim rectangle.
Trim Rectangle X1	N	The top-left corner X position of the trim rectangle in character coordinates.
Trim Rectangle Y1	N	The top-left corner Y position of the trim rectangle in character coordinates.
Trim Rectangle X2	N	The lower-right corner X position of the trim rectangle in character coordinates.
Trim Rectangle Y2	N	The lower-right corner Y position of the trim rectangle in character coordinates.
Session Presentation Space Status	N	The current status of the presentation space. The following values are possible: 1. 0: The presentation space is unlocked. 2. 4: The presentation space is busy. 3. 5: The presentation space is locked.
Session Window Handle	XXXX	Window handle of the session.

Notes:

The status of each field is updated each time the status is requested.
A new field might be added in a future version of PC/3270 or PC400.

Get System Configuration

PC/3270		*PC400*
DOS	Windows	
–	X	X

The Get System Configuration function returns the level of PC/3270 or PC400 support and other system-related values. Most of this information is for use by a service coordinator when a customer calls the IBM Support Center after receiving a system error.

```
PostMessage( hServerWnd,
             WM_DDE_REQUEST,
             hClientWnd,
              MAKELONG(cfFormat, aSYSCON) );
```

where:

cfFormat Identifies the DDE format for the data item requested. The value used is CF_TEXT.

aSYSCON
Identifies system configuration as the data item requested.

PC/3270 or PC400 Response

PC/3270 or PC400 either returns the system configuration data item in a DDE DATA message, or responds with one of these ACK messages containing status information:

WM_DDE_DATA(hData, aSYSCON)
WM_DDE_ACK(wStatus, aSYSCON)

If PC/3270 or PC400 cannot return the system configuration, a DDE ACK message will be returned with an error code in the low-order byte of the wStatus word:

WM_DDE_ACK(wStatus, aSYSCON)

Return Code	Explanation
9	A system error occurred.

Format of System Configuration information

PC/3270 or PC400 returns the system configuration as text in CF_TEXT format. The following fields are returned with the following possible values:

Fields	Returned values	Description
Version	N	The version of PC/3270 or PC400
Level	NN	The level of PC/3270 or PC400
Date	MMDDYY	The date of PC/3270 or PC400
System model base number and submodel number	XXXX	The following numbers are examples only: FC00 or FC01 = PC AT FC02 = PC XT Model 286 FC04 = PS/2 Model 50 FC05 = PS/2 Model 60 F800 or F801 = PS/2 Model 80 For more information, see *IBM Personal System/2 and Personal Computer BIOS Interface Technical Reference*.
Monitor Type	MONO, CGA, EGA, VGA, XGA	Type of the monitor
Country Code	NNNN	Country code used with 3270 or 5250 terminal session

Get System Formats

PC/3270		*PC400*
DOS	Windows	
–	X	X

The Get System Formats function returns the list of Windows clipboard formats supported by PC/3270 or PC400. The client application sends the following message to retrieve the format list supported by PC/3270 or PC400:

```
PostMessage( hServerWnd,
             WM_DDE_REQUEST,
             hClientWnd,
              MAKELONG(cfFormat, aFORMATS) );
```

where:

cfFormat Identifies the DDE format for the data item requested. The value used is CF_TEXT.

aFORMATS
Identifies formats as the data item requested.

PC/3270 or PC400 Response

PC/3270 or PC400 returns the list of supported Windows clipboard formats in CF_TEXT format in a DDE DATA message.

```
WM_DDE_DATA(hData, aFORMATS)
```

The following Windows Clipboard formats are supported by PC/3270 or PC400:

```
CF_TEXT
CF_DSPTEXT
```

If PC/3270 or PC400 cannot return the formats data item, a DDE ACK message will be returned with an error code in the low-order byte of the wStatus word:

```
WM_DDE_ACK(wStatus, aFORMATS)
```

Return Code	Explanation
9	A system error occurred.

Get System Status

PC/3270		PC400
DOS	Windows	
–	X	X

The Get System Status function returns the status of each 3270 or 5250 session that is available with the current PC/3270 or PC400 configuration. The client application sends the following message to retrieve the status data item:

```
PostMessage( hServerWnd,
             WM_DDE_REQUEST,
             hClientWnd,
              MAKELONG(cfFormat, aSTATUS) );
```

where:

cfFormat Identifies the DDE format for the data item requested. The value used is CF_TEXT.

aSTATUS
Identifies status as the data item requested.

PC/3270 or PC400 Response

PC/3270 or PC400 returns the status data item in CF_TEXT format in a DDE DATA message:

```
WM_DDE_DATA(hData, aSTATUS)
```

For each opened session, PC/3270 or PC400 returns a line of status information. Each line contains a series of fields with the following range of values:

Fields	Range of values	Description
Session ID	A, B, ..., Z	The short ID of the session.
Host Type	370, 400	The host system currently supported by PC/3270 or PC400.
Emulation Type	3270, 5250	The emulation type supported by PC/3270 or PC400.
Session Status	Closed, Invisible, Normal, Minimized, Maximized	The current status of the session's window.

If PC/3270 or PC400 cannot return the status data item, a DDE ACK message will be returned with an error code in the low-order byte of the wStatus word:

```
WM_DDE_ACK(wStatus, aSTATUS)
```

Return Code	Explanation
9	A system error occurred.

Get System SysItems

PC/3270		*PC400*
DOS	Windows	
–	X	X

PC/3270 or PC400 supports the DDE system topic so that a client application can connect to the system topic and retrieve information about PC/3270 or PC400 and the status of the sessions that PC/3270 or PC400 is managing.

The Get System SysItems function returns the list of data items available in the PC/3270 or PC400 system topic. The client application sends the following message to get the system topic data items:

```
PostMessage( hServerWnd,
             WM_DDE_REQUEST,
             hClientWnd,
              MAKELONG(cfFormat, aSYSITEMS) );
```

where:

cfFormat Identifies the DDE format for the data item requested. The value used is CF_TEXT.

aSYSITEMS
Identifies SysItems as the data item requested.

PC/3270 or PC400 Response

PC/3270 or PC400 returns the list of system topic data items in CF_TEXT format in a DDE DATA message.

```
WM_DDE_DATA(hData, aSYSITEMS)
```

The following list of data items are supported by PC/3270 or PC400:

SysItems
Topics
Status
Formats
SysCon

If PC/3270 or PC400 cannot return the system data items, a DDE ACK message will be returned with an error code in the low-order byte of the wStatus word:

```
WM_DDE_ACK(wStatus, aSYSITEMS)
```

Return Code	Explanation
9	A system error occurred.

Get System Topics

PC/3270		PC400
DOS	Windows	
–	X	X

The Get System Topics function returns the list of active DDE topics currently supported by PC/3270 or PC400. The client application sends the following message to the system topic to retrieve the list of topics that are currently active:

```
PostMessage( hServerWnd,
             WM_DDE_REQUEST,
             hClientWnd,
              MAKELONG(cfFormat, aTOPICS) );
```

where:

cfFormat Identifies the DDE format for the data item requested. The value used is CF_TEXT.

aTOPICS Identifies topics as the data item requested.

PC/3270 or PC400 Response

PC/3270 or PC400 returns the list of DDE topics in CF_TEXT format in a DDE DATA message.

```
WM_DDE_DATA(hData, aTOPICS)
```

The following list of topics are supported by PC/3270 or PC400:

System – System Topic
SessionA – Session A Topic[1]
⋮
SessionZ – Session Z Topic[1]

If PC/3270 or PC400 cannot return the list of topics, a DDE ACK message will be returned with an error code in the low-order byte of the wStatus word:

```
WM_DDE_ACK(wStatus, aTOPICS)
```

Return Code	Explanation
9	A system error occurred.

[1] The actual number of session topics supported is dependent on the number of sessions currently opened. The client program should always query the topics data item of the system topic to obtain the list of sessions currently opened.

Get Trim Rectangle

PC/3270		*PC400*
DOS	Windows	
–	X	X

The Get Trim Rectangle function returns to the client the area of the presentation space that is within the current trim rectangle. The client sends the following message to receive the trim rectangle.

```
PostMessage( hServerWnd,
             WM_DDE_REQUEST,
             hClientWnd,
              MAKELONG(cfFormat, aTRIMRECT) );
```

where:

cfFormat Identifies the format for the trim rectangle.
Valid value is CF_TEXT.

aTRIMRECT
Identifies trim rectangle as the data item requested.

PC/3270 or PC400 Response

PC/3270 or PC400 either returns trim rectangle in a DDE data message, or responds with one of these ACK messages:

WM_DDE_DATA(hData, aTRIMRECT)
WM_DDE_ACK(wStatus, aTRIMRECT)

If PC/3270 or PC400 cannot return the trim rectangle, the following status codes will be returned in the low-order byte of the wStatus word:

Return Code	Explanation
6	The specified format is not valid.
9	A system error occurred.

Initiate Session Conversation

PC/3270		*PC400*
DOS	Windows	
–	X	X

The Initiate Session Conversation function connects a client application to an available session of PC/3270 or PC400. Once a session conversation has been established, the session is reserved for exclusive use by the client until the conversation is terminated.

The client application sends the following message to initiate a DDE conversation with a session:

```
SendMessage( -1,
             WM_DDE_INITIATE,
             hClientWnd,
              MAKELONG(aIBM327 , aSessionN) );
```

where:

aIBM3270
Identifies the application atom. The string used to create atom aIBM3270 is “IBM3270”. In the PC400, the application atom is aIBM5250 and the string “IBM5250” is used to create it.

aSessionN
Identifies the topic atom. The string used to create atom aSessionN is either NULL or “Session” appended with the session ID “A”, “B”, ..., “Z”.

PC/3270 or PC400 Response

If a specific topic is selected and PC/3270 or PC400 can support a conversation with the client application, PC/3270 or PC400 acknowledges the INITIATE transaction with:

```
WM_DDE_ACK(aIBM327 , aSessionN)
```

If a topic is not selected (aSessionN = NULL), PC/3270 or PC400 responds by acknowledging all topics that are currently available:

```
WM_DDE_ACK(aIBM327 , aSystem)
WM_DDE_ACK(aIBM327 , aSessionA)
⋮
WM_DDE_ACK(aIBM327 , aSessionZ)
```

The client application selects the conversation it wishes to communicate with from the returned list of topics and terminates all other unwanted conversations.

Initiate Structured Field Conversation

PC/3270		*PC400*
DOS	Windows	
–	X	–

The Initiate Structured Field Conversation function connects a client application and a host application. This allows the applications to send data to each other and to receive data from each other.

The client sends the following command to initiate a structured field conversation:

```
SendMessage( -1,
             WM_DDE_INITIATE,
             hClientWnd,
              MAKELONG(aIBM327 , aLUN_xxxx) );
```

Where:

aIBM3270
Identifies the application atom.

aLUN_xxxx
Identifies the topic atom. The string used to create atom aLUN_xxxx is “LU” appended with the session ID “A”, “B”, ..., “Z”, appended with “_”, and appended with the user-defined string of any length.

PC/3270 response

If PC/3270 can support a structured field conversation with the client application, it will return an acknowledgment message with the following parameter:

```
WM_DDE_ACK(aIBM327 , aLUN_xxxx)
```

Initiate System Conversation

PC/3270		*PC400*
DOS	Windows	
–	X	X

The Initiate System Conversation function connects a client application to the system conversation. Only one client can be connected to the system conversation at a given time. The client sends the following command to initiate a system conversation:

```
SendMessage( -1,
             WM_DDE_INITIATE,
             hClientWnd,
              MAKELONG(aIBM327 , aSystem) );
```

where:

aIBM3270
Identifies the application atom.

aSystem Identifies the topic atom.

PC/3270 or PC400 Response

If PC/3270 or PC400 can support a system topic conversation with the client application, it returns an acknowledgment message with the following parameters:

```
WM_DDE_ACK(aIBM327 , aSystem)
```

Put Data to Presentation Space

PC/3270		PC400
DOS	Windows	
–	X	X

The Put Data to Presentation Space function sends an ASCIIZ data string to be written into the host presentation space at the location specified by the calling parameter. The client sends the following message to the session to send the string.

```
PostMessage( hServerWnd,
             WM_DDE_POKE,
             hClientWnd,
              MAKELONG(hData, aEPS) );
```

where:

hData

Identifies a handle to a Windows global memory object that contains the string to be sent to the session. The global memory object contains the following structure:

```
typedef struct tagPutString
{
 unsigned unused:13;                  //    unused
 unsigned fRelease:1;                 // Session frees memory
 unsigned freserved:2;                //    reserved
 int          cfFormat;               // Always CF_DSPTEXT
 unsigned uPSStart;                   // PS Position
 unsigned uEOFflag;                   // EOF effective switch
 char         szStringData[1];        // String Data
} PUTSTRING, FAR  lpPUTSTRING;
```

These values are valid at the uEOFflag field:

```
WC_EFFECTEOF       // The string is truncated at EOF.
WC_UNEFFECTEOF 1 // The string is not truncated at EOF.
```

aEPS Identifies the presentation space atom as the item.

PC/3270 or PC400 Response

PC/3270 or PC400 receives the string data and sends them to the presentation space, and returns a positive ACK message.

If the presentation space does not accept the string data, PC/3270 or PC400 returns a negative ACK message containing the following status codes in the low-order byte of the wStatus word:

```
WM_DDE_ACK(wStatus, aEPS)
```

Return Code	Explanation
1	PS position is not valid.
2	Length is not valid.
3	The value of EOF flag is not valid.
5	Input to the target PS was inhibited.
6	The specified format is not valid.
7	The string was truncated (successful putting).
9	A system error occurred.

Search for String

PC/3270		PC400
DOS	Windows	
–	X	X

This function allows a client application to examine the presentation space for a specified string in a specified area.

Note: The client must set the start PS position, search direction, a string to be searched, and EOF flag by using the Set Presentation Space Service Condition function before using this function.

The client sends the following message to search for the string.

```
PostMessage( hServerWnd,
             WM_DDE_REQUEST,
             hClientWnd,
              MAKELONG(cfFormat, aSTRING) );
```

where:

cfFormat Identifies the format for the search information. Valid value is CF_DSPTEXT.

aSTRING Identifies the search data item.

PC/3270 or PC400 Response

PC/3270 or PC400 returns the start position of the string in a DDE data message if the string was found in the specified area:

WM_DDE_DATA(hData, aSTRING)
WM_DDE_ACK(wStatus, aSTRING)

If PC/3270 or PC400 cannot return the start position of the string, the following status codes will be returned in the low-order byte of the wStatus word:

Return Code	Explanation
1	PS position is not valid or the string is too long.
2	The string cannot be found.
6	The specified format is not valid.
9	A system error occurred.

Structure of the Search Information

PC/3270 or PC400 returns the Search information in the following structure:

```
typedef struct tagSEARCH
{                                       //
 unsigned unused:12;                    //    unused
 unsigned fResponse:1;                  // TRUE = DDE_REQUEST response
 unsigned fRelease:1;                   // TRUE = Client frees this data
 unsigned reserved:1;                   //    reserved
 unsigned fAckReq:1;                    // TRUE = Client returns DDE_ACK
 int          cfFormat;                 // Format of Search data CF_DSPTEXT
 unsigned uFieldStart;                  // String start offset
}    SEARCH, far  lpSEARCH;
```

Send Keystrokes

PC/3270		PC400
DOS	Windows	
–	X	X

The Send Keystrokes function sends keystrokes to the connected session. The client sends the following message to the session to send keystrokes.

```
PostMessage( hServerWnd,
             WM_DDE_POKE,
             hClientWnd,
              MAKELONG(hData, aKEYS) );
```

where:

hData Identifies a handle to a Windows global memory object that contains the keystrokes to be sent to the session. The global memory object contains the following structure:

```
typedef struct tagKeystrokes
{
 unsigned unused:13;                  //    unused
 unsigned fRelease:1;                 // Session frees memory
 unsigned freserved:2;                //    reserved
 int          cfFormat;               // Always CF_DSPTEXT
 unsigned uTextType;                  // Type of keystrokes
 unsigned uRetryCount;                // Retry count 1 .. 16
 unsigned char szKeyData[1];          // Keystrokes
} KEYSTROKES, FAR  lpKEYSTROKES;
```

The following key text types are supported:

```
WC_PURETEXT       // Pure text, no AID, or included HLLAPI
                  //  commands
WC_HLLAPITEXT 1 // Text, including HLLAPI tokens
```

Note: If the keystrokes are pure text then specifying WC_PURETEXT will transfer the keystrokes to the host in the fastest possible manner. If WC_HLLAPITEXT is specified then the keystroke data can contain HLLAPI commands interspersed with the text.

aKEYS Identifies keystrokes as the item.

PC/3270 or PC400 Response

PC/3270 or PC400 receives the keystrokes and sends them to the presentation space. If the presentation space does not accept the keystrokes, a reset is sent to the presentation space and the keystrokes are sent again. This procedure continues until the presentation space accepts the keystrokes or the retry count is reached. If PC/3270 or PC400 cannot send the keystrokes to the host, PC/3270 or PC400 returns a negative ACK message containing the following status codes in the low-order byte of the wStatus word. Otherwise, PC/3270 or PC400 returns a positive ACK message signalling the keystrokes have been sent.

```
WM_DDE_ACK(wStatus, aKEYS)
```

Return Code	Explanation
1	Retry count is not valid.
2	Type of key strokes is not valid.
6	The specified format is not valid.
9	A system error occurred.

Session Execute Macro

PC/3270		PC400
DOS	Windows	
–	X	X

You can issue commands and macros with the DDE_EXECUTE function. The DDE_EXECUTE function passes command strings to PC/3270 or PC400. The command strings must conform to DDE specifications. Refer to *Microsoft Windows Software Development Kit Guide to Programming* for more information about the DDE command syntax.

The client sends the following command to issue a DDE_EXECUTE function.

```
PostMessage ( hServerWnd,
              WM_DDE_EXECUTE,
              hClientWnd,
               MAKELONG(NULL, hCommands) );
```

where:

hCommands
Identifies a handle to a Windows global memory object containing PC/3270 or PC400 commands. For a list of commands you can issue, see "Issuing Commands with the Session Execute Macro Function."

PC/3270 or PC400 Response

If PC/3270 or PC400 can process the command string, PC/3270 or PC400 returns an ACK message containing positive status information to the client. If PC/3270 or PC400 cannot perform the command string, PC/3270 or PC400 returns an ACK message containing this error code in the low-order word of the wStatus word:

Return Code	Explanation
9	A system error occurred.

Issuing Commands with the Session Execute Macro Function

You can issue the following commands with the Session Execute Macro function:

WINDOW
KEYBOARD
SEND
RECEIVE
SENDKEY
WAIT

Use a separate DDE_EXECUTE message for each command.

Notes:

Enclose values that contain nonalphanumeric characters or blanks in double quotation marks ("*value value*").
To include a double quotation mark within a string, type it twice (for example, "This is a double quotation mark:""").
The maximum length for any command is 255 characters.

WINDOW Command

[WINDOW(action, "name"])]

Performs window actions, where:

action = HIDE|RESTORE|MAXIMIZE|MINIMIZE|
SHOW|CNGNAME
name = LT name or Switch List Entry name

Note: *name* should be specified when CNGNAME is specified at *action*. If *name* is a NULL string, the default caption will be displayed.

KEYBOARD Command

[KEYBOARD(action)]

Enables or disables the session keyboard, including the mouse, where:

action = LOCK|UNLOCK

SEND Command

[SEND("pcfilename ","hostfilename ","options ")]

Sends files to the host, where:

```
 pcfilename  = [path ]filename [.ext ]
hostfilename  =
   For VM system:
     filename filetype  [filemode ]
   For MVS system:
      ']filename  [(membername)][']
   For CICS system:
     filename  or  _
   For OS/4  system:
     library name  filename  member name
```

options includes any combination of the following file transfer options: MVS, VM, CICS, QUIET, OS/400, and emulation-specific transfer options, separated by spaces.

Refer to *Personal Communications Version 4.1 Reference for DOS (Full-Function Mode)* or *Personal Communications Version 4.1 Reference for Windows* for more information about the transfer options.

RECEIVE Command

[RECEIVE("pcfilename","hostfilename ","options ")]

Receives files from the host, where:

```
 pcfilename  = [path ]filename [.ext ]
hostfilename  =
   For VM system:
     filename filetype  [filemode ]
   For MVS system:
      ']filename  [(membername)][']
   For CICS system:
     filename  or  _
   For OS/4  system:
     library name  filename  member name
```

options includes any combination of the following file transfer options: MVS, VM, CICS, QUIET, OS/400, and emulation-specific transfer options, separated by spaces.

Refer to *Personal Communications Version 4.1 Reference for DOS (Full-Function Mode)* or *Personal Communications Version 4.1 Reference for Windows* for more information about the transfer options.

SENDKEY Command

[SENDKEY(token,token)]

Sends keystrokes to PC/3270 or PC400, where:

token = text string | command | macro macroname

Notes:

Text strings are enclosed in double quotation marks.
Macros are prefixed with “macro”.
The argument string for SENDKEY must be 255 characters or less in size.
The following commands are supported.

Table 17 (Page 1 of 3). SENDKEY Command List

Command Name	Token	PC/3270	PC400
Alternate Cursor	[alt cursor]	X	X
Alternate Viewing Mode	[alt view]	X	X
Attention	[sys attn]	X	X
Backspace	[backspace]	X	X
Back Tab	[backtab]	X	X
Backtab Word	[backtab word]	X	X
Character Advance	[character advance]	–	X
Character Backspace	[backspace valid]	–	X
Clear Screen	[clear]	X	X
Clicker	[click]	X	X
Color Blue	[blue]	X	–
Color Field Inherit	[field color]	X	–
Color Green	[green]	X	–
Color Pink	[pink]	X	–
Color Red	[red]	X	–
Color Turquoise	[turquoise]	X	–
Color White	[white]	X	–
Color Yellow	[yellow]	X	–
Cursor Blink	[cursor blink]	X	X
Cursor Down	[down]	X	X
Cursor Left	[left]	X	X
Cursor Right	[right]	X	X
Cursor Select	[cursor select]	X	X
Cursor Up	[up]	X	X
Delete Character	[delete char]	X	X
Delete Word	[delete word]	X	X
Device Cancel	[device cancel]	X	X
Dup Field	[dup]	X	X
Edit Clear	[edit-clear]	X	X
Edit Copy	[edit-copy]	X	X
Edit Cut	[edit-cut]	X	X

Table 17 (Page 2 of 3). SENDKEY Command List

Command Name	Token	PC/3270	PC400
Edit Paste	[edit-paste]	X	X
Edit Undo	[edit-undo]	X	X
End Field	[end field]	X	X
Enter	[enter]	X	X
Erase EOF	[erase eof]	X	X
Erase Field	[erase field]	X	–
Erase Input	[erase input]	X	X
Fast Cursor Down	[fast down]	X	X
Fast Cursor Left	[fast left]	X	X
Fast Cursor Right	[fast right]	X	X
Fast Cursor Up	[fast up]	X	X
Field Exit	[field exit]	–	X
Field Mark	[field mark]	X	X
Field +	[field +]	–	X
Field –	[field -]	–	X
Graphic Cursor	[+cr]	X	–
Help	[help]	X	X
Highlighting Field Inherit	[field hilight]	X	–
Highlighting Reverse	[reverse]	X	–
Highlighting Underscore	[underscore]	X	–
Home	[home]	X	X
Host Print	[host print]	X	–
Input	[input]	X	X
Input nondisplay	[input nd]	X	X
Insert Toggle	[insert]	X	X
Lower case	[to lower]	X	–
Mark Down	[mark down]	X	X
Mark Left	[mark left]	X	X
Mark Right	[mark right]	X	X
Mark Up	[mark up]	X	X
Move Mark Down	[move down]	X	X
Move Mark Left	[move left]	X	X
Move Mark Right	[move right]	X	X
Move Mark Up	[move up]	X	X
New Line	[newline]	X	X
Next Page	[page down]	–	X
Pause 1 second	[pause]	X	X
Previous Page	[page up]	–	X
Print Screen	[local copy]	X	X
Program Attention Key 1	[pa1]	X	–
Program Attention Key 2	[pa2]	X	–
Program Attention Key 3	[pa3]	X	–

Table 17 (Page 3 of 3). SENDKEY Command List

Command Name	Token	PC/3270	PC400
Program Function Key 1	[pf1]	X	X
⋮	⋮	⋮	⋮
Program Function Key 24	[pf24]	X	X
Quit	[quit]	X	X
Reset	[reset]	X	X
Response Time Monitor	[rtm]	X	–
Roll Down	[roll down]	–	X
Roll Up	[roll up]	–	X
Rubout	[rubout]	X	X
Rule	[rule]	X	X
SO/SI Display	[so si]	X	X
SO/SI Generate	[so si generate]	–	X
System Request	[sys req]	X	X
Tab Field	[tab field]	X	X
Tab Word	[tab word]	X	X
Test	[test request]	–	X
Unmark	[unmark]	X	X
Upper case	[to upper]	X	–
Upper/Lower Change	[to other]	X	–
Wait for bind	[wait app]	X	X
Wait for System	[wait sys]	X	X
Wait transition	[wait trn]	X	X
Wait while input inh.	[wait inp inh]	X	X
Window Relocation 1	[view 1]	X	X
⋮	⋮	⋮	⋮
Window Relocation 8	[view 8]	X	X

WAIT Command

[WAIT("[time out][wait condition]")]

Waits until the timeout expires or the wait condition the client specified occurs. For this command, the client has to set at least one option, where:

time out (optional)
: If the client sets a timeout value in the command statements, the following units are available in the wait statement.

  msec
  millisecond
  milliseconds
  sec
  second
  seconds
  minute
  minutes
  hour
  hours

wait condition (optional)

For the wait condition option, the client can select the following options:

while cursor at (cursor row, cursor column)
: While the cursor is at (cursor row, cursor column), it keeps waiting.

while “string”
: While the “string” is somewhere on the screen, it keeps waiting.

while “string” at (cursor row, cursor column)
: While the “string” is at (cursor row, cursor column) on the screen, it keeps waiting.

until cursor at (cursor row, cursor column)
: Until the cursor moves to (cursor row, cursor column), it keeps waiting.

until “string”
: Until the “string” is displayed somewhere on the screen, it keeps waiting.

until “string” at (cursor row, cursor column)
: Until the “string” is displayed at (cursor row, cursor column), it keeps waiting.

Examples:

1. To wait 10 seconds

   [WAIT("1 seconds")]

2. To wait while "ABCDEF" is displayed at (2,9) on the screen

   [WAIT("while ""ABCDEF"" at (2,9)")]

3. To wait until "ABCDEF" is displayed at (2,9) on the screen, or after 8 seconds

   [WAIT("8 seconds until ""ABCDEF"" at (2,9)")]

Set Cursor Position

PC/3270		PC400
DOS	Windows	
–	X	X

The Set Cursor Position function allows the client application to set the cursor position in the session window.

```
PostMessage( hServerWnd,
             WM_DDE_POKE,
             hClientWnd,
              MAKELONG(hData, aSETCURSOR) );
```

where:

hData

Identifies a handle to a Windows global memory object that contains the cursor positioning information in the following structure:

```
typedef struct tagSETCURSOR
{
 unsigned unused:13;                //    unused
 unsigned fRelease:1;               // TRUE = Session frees memory
 unsigned freserved:2;              //    reserved
 int          cfFormat;               // Always CF_DSPTEXT
 unsigned uSetCursorType;           // Cursor Set Type
 unsigned uSetCursor1;              // Cursor Row or PS Offset
 unsigned uSetCursor2;              // Cursor Col
} SETCURSOR, FAR  lpSETCURSOR;
```

PC/3270 or PC400 supports two ways to set the cursor position:

PS Offset (uSetCursorType = 0)
Row/Column number (uSetCursorType = 1)

The application specifies which method by setting the uSetCursorType field to the appropriate value, followed by setting the two other fields uSetCursor1 and uSetCursor2 to their appropriate values as follows:

uSetCursorType = 0 offset

- uSetCursor1: 0 ... (PSsize – 1)

uSetCursorType = 1 row/col

- uSetCursor1: 0 ... (PSrows – 1)
- uSetCursor2: 0 ... (PScols – 1)

aSETCURSOR

Identifies cursor position as the item.

PC/3270 or PC400 Response

PC/3270 or PC400 receives the cursor information and moves the cursor to the specified position in the PS. If the cursor is positioned successfully, PC/3270 or PC400 returns a positive ACK message to the client application. Otherwise, a negative ACK message is returned with the following error codes in the low-order byte of the wStatus word.

```
WM_DDE_ACK(wStatus, aSETCURSOR)
```

Return Code	Explanation
1	Cursor set type is not valid. Must be 0 or 1.
2	Cursor PS offset is not valid. Must be 0 ... (PSsize – 1).
3	Cursor row value is not valid. Must be 0 ... (PSrows – 1).
4	Cursor column value is not valid. Must be 0 ... (PScols – 1).
6	The specified format is not valid.
9	A system error occurred.

Set Mouse Intercept Condition

PC/3270		*PC400*
DOS	Windows	
–	X	X

This function specifies the mouse input to be intercepted. The client sends the following command to set the mouse event to be intercepted.

```
PostMessage( hServerWnd,
             WM_DDE_POKE,
             hClientWnd,
              MAKELONG(hData, aMOUSE) );
```

where:

hData Identifies a handle to a Windows global memory object that specifies the condition of intercepting the mouse input.

If the format is CF_TEXT, the client program sends the condition in the following structure:

```
typedef struct tagSETMOUSE_CF_TEXT
{
 unsigned          unused:12,                //
 unsigned          fRelease:1,              //
 unsigned          fReserved:3;             //
 int               cfFormat;                 // Always CF_TEXT
 unsigned char Condition[1]                 //
} SETMOUSE_CF_TEXT, FAR  lpSETMOUSE_CF_TEXT;
```

The following table shows the parameters' values:

Parameter Name	Meaning	Value
Condition	Condition of intercepting the mouse input	A string terminated with '\0', consists of the constants defined as follows in any order: **L** Enable intercepting the left button **l** Disable intercepting the left button **R** Enable intercepting the right button **r** Disable intercepting the right button **M** Enable intercepting the middle button **m** Disable intercepting the middle button **S** Enable intercepting a single click **s** Disable intercepting a single click **D** Enable intercepting a double click **d** Disable intercepting a double click **T** Retrieve the pointed string **t** Do not retrieve the pointed string

If the format is CF_DSPTEXT, the client program sends the condition in the following structure:

```
typedef struct tagSETMOUSE_CF_DSPTEXT
{
 unsigned        unused:12,              //
 unsigned        fRelease:1,             //
 unsigned        fReserved:3;            //
 int             cfFormat;               // Always CF_DSPTEXT
 BOOL              bLeftButton;          //
 BOOL              bRightButton;         //
 BOOL              bMiddleButton;        //
 BOOL              bSingleClick;         //
 BOOL              bDoubleClick;         //
 BOOL              bRetrieveString;      //
} SETMOUSE_CF_DSPTEXT, FAR  lpSETMOUSE_CF_DSPTEXT;
```

The following table shows the values in the parameters:

Parameter Name	Meaning	Value	
bLeftButton	Enable or disable interception of the left mouse button	**True** **False**	Enable intercepting the left button Disable intercepting the left button
bRightButton	Enable or disable interception of the right mouse button	**True** **False**	Enable intercepting the right button Disable intercepting the right button
bMiddleButton	Enable or disable interception of the middle mouse button	**True** **False**	Enable intercepting the middle button Disable intercepting the middle button
bSingleClick	Enable or disable interception of the single click	**True** **False**	Enable intercepting the single click Disable intercepting the single click
bDoubleClick	Enable or disable interception of the double click	**True** **False**	Enable intercepting the double click Disable intercepting the double click
bRetrieveString	Retrieve or do not retrieve the pointed string	**True** **False**	Retrieve the pointed string Do not retrieve the pointed string

aMOUSE Identifies Mouse atom as the item.

PC/3270 or PC400 Response

When receiving the Set Mouse Intercept Condition request, PC/3270 or PC400 returns an ACK message if it can set the intercept condition to the specified status. Otherwise, a negative ACK message is returned to the client with one of the following return codes in the low-order byte of the wStatus field:

WM_DDE_ACK(wStatus, aMOUSE)

Return Code	Explanation
2	A character in Condition parameter is not valid.
6	The specified format is not valid.
9	A system error occurred.

Set Presentation Space Service Condition

PC/3270		PC400
DOS	Windows	
–	X	X

The Set Presentation Space Service Condition function sets the condition for using the following functions:

Get Partial Presentation Space
Find Field
Search for String

The client application sets the condition by calling this function such as:

Start PS position
PS length
EOF flag
Search direction
ASCIIZ string

The client must specify the Set Presentation Space Service Condition function before invoking the functions listed above. The conditions set by this function remain in effect until the next Set Presentation Space Service Condition function is called. The client sends the following message to set the condition:

```
PostMessage( hServerWnd,
             WM_DDE_POKE,
             hClientWnd,
              MAKELONG(hData, aEPSCOND) );
```

where:

hData

Identifies a handle to a Windows global memory object containing:

```
typedef struct tagPSSERVCOND
{
 unsigned     unused:13,                  //
 unsigned     fRelease:1,                //
 unsigned     fReserved:2;               //
 int          cfFormat;                   // Always CF_DSPTEXT
 unsigned     uPSStart;                   // PS Position
 unsigned     uPSLength;                   // Length of PS
 unsigned     uSearchDir;                 // Direction for search
 unsigned     uEOFflag;                    // EOF effective switch
 char         szTargetString[1];         // Target String
} PSSERVCOND, FAR  lpPSSERVCOND;
```

The following values are valid at the uSearchDir field:

```
WC_SRCHFRWD       // Search forward.
WC_SRCHBKWD      1 // Search backward.
```

The following values are valid at the uEOFflag field:

```
WC_UNEFFECTEOF   // The string is not truncated at EOF.
WC_EFFECTEOF     1 // The string is truncated at EOF.
```

aEPSCOND

Identifies the item for the Set Presentation Space Service Condition function.

PC/3270 or PC400 Response

If PC/3270 or PC400 can perform the Set Presentation Space Service Condition function, then PC/3270 or PC400 returns an ACK message:

WM_DDE_ACK(wStatus, aEPSCOND)

If PC/3270 or PC400 cannot perform the Set Presentation Space Service Condition function, then PC/3270 or PC400 returns a negative ACK message containing the following return codes in the low-order byte of wStatus:

Return Code	Explanation
1	PS position is not valid.
2	Length is not valid.
3	The value of EOF flag is not valid.
4	The value of Search Direction is not valid.
6	The specified format is not valid.
9	A system error occurred.

Set Session Advise Condition

PC/3270		*PC400*
DOS	Windows	
–	X	X

This function sets the condition for the DDE_ADVISE of the Start Session Advise function. The client can specify a search string and a region of the screen. When the advise condition is met, the server notifies the client of the condition according to the options specified by the Start Session Advise function.

Note: The client must specify the Set Session Advise Condition function before invoking Start Session Advise. If the advise condition is set after the Start Session Advise function is started, the advise condition will be ignored and the client will receive a negative ACK message. See “Start Session Advise” on page 249 for more information about starting the advise.

The client sends the following message to set the advise condition.

```
PostMessage( hServerWnd,
             WM_DDE_POKE,
             hClientWnd,
              MAKELONG(hData, aPSCOND) );
```

where:

hData Identifies a handle to a Windows global memory object containing:

```
typedef struct tagSEARCHDATA
{
 unsigned     unused:13,                  //
 unsigned     fRelease:1,                //
 unsigned     fReserved:2;               //
 int          cfFormat;                   // Always CF_DSPTEXT
 WORD           uPSStart;                    // PS Position of string
 WORD           uPSLength                     // Length of String
 BOOL         bCaseSensitive;            // Case Sensitive TRUE=YES
  char        SearchString[1];          //  Search  String
} SEARCHDATA, FAR  lpSEARCHDATA;
```

aPSCOND

Identifies the item for the Set Session Advise Condition function.

PC/3270 or PC400 Response

If PC/3270 or PC400 can perform the Set Session Advise Condition function, PC/3270 or PC400 returns this ACK message:

WM_DDE_ACK(wStatus, aPSCOND)

If PC/3270 or PC400 cannot perform the Set Session Advise Condition function, then PC/3270 or PC400 returns an negative ACK message containing the following return codes in the low-order byte of wStatus:

Return Code	Explanation
1	Advise is already active.
2	Advise condition is already active.
3	PS position is not valid.
4	String length is not valid.
6	The specified format is not valid.
9	A system error occurred.

Set Structured Field Service Condition

PC/3270		PC400
DOS	Windows	
–	X	–

The Set Structured Field Service Condition function passes the Query Reply data provided by the client application.

Note: The client must call the Set Structured Field Service Condition function before invoking the Start Read SF function or the Write SF function.

The client sends the following message to set the condition.

```
PostMessage( hServerWnd,
             WM_DDE_POKE,
             hClientWnd,
              MAKELONG(hData, aSFCOND) );
```

where:

hData Identifies a handle to a Windows global memory object containing:

```
typedef struct tagSFSERVCOND
{
 unsigned     unused:12,              //
 unsigned     fRelease:1,             //
 unsigned     fReserved:3;            //
 int          cfFormat;               // Always CF_DSPTEXT
 WORD           uBufferSize;          // Buffer size of Read SF
 WORD           uQRLength;            // Length of Query Reply data
 char         szQueryReply[1];        // Query Reply data
} SFSERVCOND, FAR  lpSFSERVCOND;
```

aSFCOND

Identifies the item for the Set Structured Field Service Condition function.

PC/3270 response

PC/3270 checks the Query Reply ID and Type (not DOID) and the length. If they are valid, then PC/3270 returns an ACK message:

```
WM_DDE_ACK(wStatus, aSFCOND)
```

If PC/3270 cannot perform the Set Structured Field Service Condition function, then PC/3270 returns a negative ACK message containing the following return codes in the low-order byte of wStatus:

Return Code	Explanation
1	PS SF ID is not valid.
2	Length is not valid.
3	One DDM base type is already connected to this session.
4	Structured Field Service Condition is already set.
6	The specified format is not valid.
9	A system error occurred.

Start Close Intercept

PC/3270		PC400
DOS	Windows	
–	X	X

The Start Close Intercept function allows a client application to intercept close requests generated when a user selects the close option from the emulator session window. This function intercepts the close request and discards it until the Stop Close Intercept function is requested. After using this function, the client receives DATA messages notifying it that close requests occurred (CLOSE).

The client sends the following command to begin a session advise.

```
PostMessage( hServerWnd,
             WM_DDE_ADVISE,
             hClientWnd,
              MAKELONG(hOptions, aCLOSE) );
```

where:

hOptions Is a handle to a Windows global memory object, with the following structure:

```
typedef struct tagOPTIONS
{
 unsigned reserved:14;          //    reserved
 unsigned fDeferUpd:1;           // Send notification only
 unsigned fAckReq:1;              // Client will ACK all notices
 WORD        cfFormat;                 // Clipboard format to use
}   OPTIONS, FAR  lpOPTIONS;
```

If the value of fDeferUpd is 1, DDE Data messages will be sent to the client application with the hData set to NULL. The client must then issue a DDE REQUEST to request the data item.

If the value of fAckReq is 1, the server does not notify the client of further close requests until the server receives an ACK message from the client in response to any previous notification.

The cfFormat field specifies the format to send the close request. (Must be CF_DSPTEXT.)

aCLOSE Identified close intercept as the item.

PC/3270 or PC400 Response

PC/3270 or PC400 receives the Start Close Intercept and returns an ACK message if it can start the intercept. Otherwise a negative ACK message will be returned to the client with the following return codes in the low-order byte of the wStatus field:

```
WM_DDE_ACK(wStatus, aCLOSE)
```

Return Code	Explanation
1	Close Intercept is already working.
6	The specified format is not valid.
9	A system error occurred.

Once the intercept starts, the client receives DATA messages notifying it that the close request is intercepted:

WM_DDE_DATA(hData, aCLOSE)

where:

hData Identifies a handle to a Windows global memory object containing:

```
typedef struct tagCLOSEREQ
}
 unsigned     unused:12,            //     unused
 unsigned     fResponse:1,          // TRUE = DD_REQUEST response
 unsigned     fRelease:1,           // TRUE = Client releases memory
 unsigned     reserved:1,           //     reserved
 unsigned     fAckReq:1,            // TRUE = DDE_ACK is required
  int         cfFormat;             //  Always  CF_DSPTEXT
 WORD            uCloseReqCount;    // Counter of the Close Requests
} CLOSEREQ, FAR  lpCLOSEREQ;
```

The DATA messages continue until a Stop Close Intercept message is sent to PC/3270 or PC400.

Start Keystroke Intercept

PC/3270		PC400
DOS	Windows	
–	X	X

The Start Keystroke Intercept function allows a client application to filter any keystrokes sent to a session by a terminal operator. After a call to this function, keystrokes are intercepted, and the client receives them (KEYS).

The client sends the following command to begin intercept.

```
PostMessage( hServerWnd,
             WM_DDE_ADVISE,
             hClientWnd,
              MAKELONG(hOptions, aKEYS) );
```

where:

hOptions Is a handle to a Windows global memory object, with the following structure:

```
typedef struct tagOPTIONS
}
 unsigned reserved:14;          // Reserved
 unsigned fDeferUpd:1;           // Send notification only
 unsigned fAckReq:1;             // Client will ACK all notices
 WORD       cfFormat;                 // Clipboard format to use
}   OPTIONS, FAR  lpOPTIONS;
```

If the value of fDeferUpd is 1, DDE Data messages are sent to the client application with the hData set to NULL. The client then issues a DDE REQUEST to request the data item.

If the value of fAckReq is 1, the server does not notify the client of further keystrokes until the server receives an ACK message from the client in response to any previous keystrokes notification.

The cfFormat field specifies the format to send the keystrokes when the keystroke is sent by a terminal operator. (Must be CF_DSPTEXT.)

aKEYS Identified keystrokes as the item.

PC/3270 or PC400 Response

PC/3270 or PC400 receives the Start Keystroke Intercept and returns an ACK message if it can start the intercept. Otherwise a negative ACK message is returned to the client with the following return codes in the low-order byte of the wStatus field:

```
WM_DDE_ACK(wStatus, aKEYS)
```

Return Code	Explanation
1	Keystroke Intercept is already started.
6	The specified format is not valid.
9	A system error occurred.

Once the intercept has started, the client receives DATA messages notifying it that the keystroke is intercepted:

```
WM_DDE_DATA(hData, aKEYS)
```

The DATA messages continue until a Stop Keystroke Intercept message is sent to PC/3270 or PC400. The format of the data item will be the same format as if the client requested the data item via a DDE_REQUEST.

Start Mouse Input Intercept

PC/3270		*PC400*
DOS	Windows	
–	X	X

The Start Mouse Input Intercept function allows a client application to intercept mouse input when a terminal operator press the mouse button on emulator session window. After calling this function, the client receives DATA messages that include the PS position where mouse input occurred.

The client sends the following command to begin to intercept the mouse input.

```
PostMessage( hServerWnd,
             WM_DDE_ADVISE,
             hClientWnd,
              MAKELONG(hOptions, aMOUSE) );
```

where:

hOptions Is a handle to a Windows global memory object, with the following structure:

```
typedef struct tagOPTIONS
{
 unsigned reserved:14;              // Reserved
 unsigned fDeferUpd:1;               // Send notification only
 unsigned fAckReq:1;                  // Client will ACK all notices
 WORD         cfFormat;                    // Clipboard format to use
}    OPTIONS, FAR  lpOPTIONS;
```

If the value of fDeferUpd is 1, DDE Data messages will be sent to the client application with the hData set to NULL. The client must then issue a DDE REQUEST to request the data item.

If the value of fAckReq is 1, the server does not notify the client of further structured field data until the server receives an ACK message from the client in response to any previous notification.

The cfFormat field specifies the format to send the data item has been updated.

aMOUSE Identified MOUSE as the item.

PC/3270 or PC400 response

PC/3270 or PC400 receives the Start Mouse Input Intercept and returns an ACK message if it can start this function. Otherwise a negative ACK message is returned to the client with the following return codes in the low-order byte of the wStatus field:

WM_DDE_ACK(wStatus, aMOUSE)

Return Code	Explanation
1	Mouse Input Intercept has been already started.
6	The specified format is not valid.
9	A system error occurred.

Once the Mouse Input Intercept starts, the client receives DATA messages of the structured field:

WM_DDE_DATA(hData, aMOUSE)

where:

hData

If the format is CF_TEXT, PC/3270 or PC400 returns the mouse input information in the following format:

```
typedef struct tagMOUSE_CF_TEXT
{
 unsigned          unused:12,          //     Unused
  unsigned         fRespons:1,        // TRUE = DDE_REQUEST response
  unsigned         fRelease:1,        // TRUE = Client frees this data
  unsigned         reserved:1,        //      Reserved
 unsigned          fAckReq:1;           // TRUE = Client returns DDE_ACK
 int               cfFormat;            // CF_TEXT
 unsigned char PSPos[4];              // PS position
 unsigned char Tab1[1];               // TAB character
 unsigned char PSRowPos[4];           // PS row position
 unsigned char Tab2[1];               // TAB character
 unsigned char PSColPos[4];           // PS columns position
 unsigned char Tab3[1];               // TAB character
 unsigned char PSSize[4];             // Size of the PS
 unsigned char Tab4[1];               // TAB character
 unsigned char PSRows[4];             // PS number of rows
 unsigned char Tab5[1];               // TAB character
 unsigned char PSCols[4];             // PS number of columns
 unsigned char Tab6[1];               // TAB character
 unsigned char ButtonType[1];         // Pressed button type
 unsigned char Tab7[1];               // TAB character
 unsigned char ClickType[1];          // Click type
 unsigned char Tab8[1];               // TAB character
 unsigned char ClickString[1];    // Retrieved string
} MOUSE_CF_TEXT, FAR  lpMOUSE_CF_TEXT;
```

The following table shows the values in the parameters:

Parameter Name	Meaning	Value
PSPos	PS offset of the position where mouse was clicked	0 ... (PSSize – 1)
PSRowPos	Row number of the position where mouse was clicked	0 ... (PSRows – 1)
PSColPos	Column number of the position where mouse was clicked	0 ... (PSCols – 1)
PSSize	Presentation space size	
PSRows	Number of presentation space rows	
PSCols	Number of presentation space columns	
ButtonType	Type of clicked mouse button	**L** Left button **M** Middle button **R** Right button
ClickType	Type of clicking	**S** Single click **D** Double click
ClickString	Retrieved string to which the mouse pointed	A character string terminated with a '\0'
Tab1–8	A tab character for delimiter	'\t'

If the format is CF_DSPTEXT, PC/3270 or PC400 returns the mouse input information in the following format:

```
typedef struct tagMOUSE_CF_DSPTEXT
{
 unsigned        unused:12,          //     Unused
 unsigned        fRespons:1,         // TRUE = DDE_REQUEST response
 unsigned        fRelease:1,         // TRUE = client frees the storage
 unsigned        reserved:1,         //     Reserved
 unsigned        fAckReq:1;          // TRUE = client returns DDE_ACK
 int             cfFormat;           // CF_DSPTEXT
 unsigned        uPSPos;             // PS position
 unsigned        uPSRowPos;          // PS row position
 unsigned        uPSColPos;          // PS column position
 unsigned        uPSSize;            // Size of the presentation space
 unsigned        uPSRows;            // PS number of rows
 unsigned        uPSCols;            // PS number of columns
 unsigned        uButtonType;        // Pressed button type
 unsigned        uClickType;         // Click type
 unsigned char szClickString[1];   // Retrieved string
} MOUSE_CF_DSPTEXT, FAR  lpMOUSE_CF_DSPTEXT;
```

The following table shows the values in the parameters:

Parameter Name	Meaning	Value
uPSPos	PS offset of the position where mouse was clicked	0 ... (uPSSize – 1)
uPSRowPos	Row number of the position where mouse was clicked	0 ... (uPSRows – 1)
uPSColPos	Column number of the position where mouse was clicked	0 ... (uPSCols – 1)
uPSSize	Size of the presentation space	
uPSRows	Number of rows of the presentation space	
uPSCols	Number of columns of the presentation space	
uButtonType	Type of the clicked mouse button	**0x0001** Left button **0x0002** Middle button **0x0003** Right button
uClickType	Type of clicking	**0x0001** Single click **0x0002** Double click
szClickString	Retrieved string to which the mouse pointed	A character string terminated with a '\0'

The DATA messages continue until a Stop Mouse Input Intercept message is sent to PC/3270 or PC400.

Start Read SF

PC/3270		PC400
DOS	Windows	
–	X	–

The Start Read SF function allows a client application to read structured field data from the host application. After using this function, the client receives DATA messages notifying it that close requests occurred.

Note: Before using this function, the client must call the Set Structured Field Service Condition function to pass the Query Reply data to the server.

The client sends the following command to begin a Read SF.

```
PostMessage( hServerWnd,
             WM_DDE_ADVISE,
             hClientWnd,
              MAKELONG(hOptions, aSF) );
```

where:

hOptions Is a handle to a Windows global memory object, with the following structure:

```
typedef struct tagOPTIONS
{
 unsigned reserved:14;              // Reserved
 unsigned fDeferUpd:1;               // Send notification only
 unsigned fAckReq:1;                  // Client will ACK all notices
 WORD         cfFormat;                    // Clipboard format to use
}    OPTIONS, FAR  lpOPTIONS;
```

If the value of fDeferUpd is 1, DDE Data messages will be sent to the client application with the hData set to NULL. The client must then issue a DDE REQUEST to request the data item.

If the value of fAckReq is 1, the server does not notify the client of further structured field data until the server receives an ACK message from the client in response to any previous notification.

The cfFormat field specifies the format to send the structured field data. (It must be CF_DSPTEXT.)

aSF Identified structured field as the item.

PC/3270 response

PC/3270 receives the Start Read SF and returns an ACK message if it can start the Read SF. Otherwise a negative ACK message will be returned to the client with the following return codes in the low-order byte of the wStatus field:

WM_DDE_ACK(wStatus, aSF)

Return Code	Explanation
1	Read SF is already started.
3	No prior Set Structured Field Service Condition function was called.
6	The specified format is not valid.
9	A system error occurred.

Once the Read SF has started, the client receives DATA messages of the structured field:

WM_DDE_DATA(hData, aSF)

where:

hData Identifies a handle to a Windows global memory object containing:

```
typedef struct tagSFDATA
{
 unsigned     unused:12,                //
 unsigned     fRelease:1,              //
 unsigned     fReserved:3;              //
 int          cfFormat;                 // Always CF_DSPTEXT
 WORD           uSFLength;                    // Length of SF data
 char         szSFData[1];              // SF data
} SFDATA, FAR  lpSFDATA;
```

The DATA messages continue until a Stop Read SF message is sent to PC/3270.

Start Session Advise

PC/3270		PC400
DOS	Windows	
–	X	X

The Start Session Advise function establishes a link between the PC/3270 or PC400 session and the client. This lets the client receive updates of the presentation space (PS), the operator information area (OIA), or the trim rectangle (TRIMRECT) when the data item is updated.

Note: If the client application needs conditional notification when the presentation space is updated, set an advise condition prior to invoking the advise function for the presentation space. See “Set Session Advise Condition” on page 236.

The client sends the following command to begin a session advise.

```
PostMessage( hServerWnd,
             WM_DDE_ADVISE,
             hClientWnd,
              MAKELONG(hOptions, aItem) );
```

where:

hOptions Is a handle to a Windows global memory object. This is the structure:

```
typedef struct tagOPTIONS
{
 unsigned reserved:14;              // Reserved
 unsigned fDeferUpd:1;               // Send notification only
 unsigned fAckReq:1;                  // Client will ACK all notices
 WORD        cfFormat;                    // Clipboard format to use
}    OPTIONS, FAR  lpOPTIONS;
```

If the value of fDeferUpd is 1, DDE Data messages are sent to the client application with the hData set to NULL. The client must then issue a DDE REQUEST to request the data item.

If the value of fAckReq is 1, the server does not notify the client of further changes to the data item until the server receives an ACK message from the client in response to any previous update notification.

The cfFormat field specifies the format to send the data item when the item has been updated.

aItem Specifies the item of information being requested; in this case, the value can be PS, OIA, or TRIMRECT.

PC/3270 or PC400 Response

PC/3270 or PC400 receives the Start Session Advise and returns an ACK message if it can start the advise. Otherwise, a negative ACK message will be returned to the client with the following return codes in the low-order byte of the wStatus field:

WM_DDE_ACK(wStatus, aItem)

Return Code	Explanation
1	Advise already active for data item.
6	Advise parameter not valid.
9	A system error occurred.

Once the advise has started, the client receives DATA messages notifying it that the data item (PS, OIA, or TRIMRECT) has changed:

WM_DDE_DATA(hData, aItem)

The DATA messages continue until a Stop Session Advise message is sent to PC/3270 or PC400. The format of the data item will be the same format as if the client requested the data item via a DDE_REQUEST.

Stop Close Intercept

PC/3270		*PC400*
DOS	Windows	
–	X	X

The Stop Close Intercept function ends a client application's ability to intercept close request. The client sends the following command to perform the Stop Close Intercept function.

```
PostMessage( hServerWnd,
             WM_DDE_UNADVISE,
             hClientWnd,
              MAKELONG(NULL, aCLOSE) );
```

where:

aCLOSE Identified close intercept as the item.

PC/3270 or PC400 Response

If PC/3270 or PC400 can perform the DDE_UNADVISE, PC/3270 or PC400 returns an ACK message containing positive status information to the client:

```
WM_DDE_ACK(wStatus, aCLOSE)
```

If PC/3270 or PC400 cannot perform the DDE_UNADVISE, PC/3270 or PC400 returns an ACK message containing negative status information and the following return codes in the low-order byte of the wStatus word:

Return Code	Explanation
1	Advise has not started yet.
9	A system error occurred.

Stop Keystroke Intercept

PC/3270		*PC400*
DOS	Windows	
–	X	X

The Stop Keystroke Intercept function ends a client application's ability to intercept keystrokes. The client sends the following command to perform the Stop Keystroke Intercept function.

```
PostMessage( hServerWnd,
             WM_DDE_UNADVISE,
             hClientWnd,
              MAKELONG(NULL, aKEYS) );
```

where:

aKEYS Identified keystrokes as the item.

PC/3270 or PC400 Response

If PC/3270 or PC400 can perform the DDE_UNADVISE, PC/3270 or PC400 returns an ACK message containing positive status information to the client:

```
WM_DDE_ACK(wStatus, aKEYS)
```

If PC/3270 or PC400 cannot perform the DDE_UNADVISE, PC/3270 or PC400 returns an ACK message containing negative status information and the following return codes in the low-order byte of the wStatus word.

Return Code	Explanation
1	Advise has not started yet.
9	A system error occurred.

Stop Mouse Input Intercept

PC/3270		*PC400*
DOS	Windows	
–	X	X

The Stop Mouse Input Intercept function ends a client application's ability to intercept mouse input.

The client sends the following command to perform the Stop Mouse Input Intercept function.

```
PostMessage( hServerWnd,
             WM_DDE_UNADVISE,
             hClientWnd,
              MAKELONG(NULL, aMOUSE) );
```

where:

aMOUSE Identified MOUSE as the item.

PC/3270 or PC400 Response

If PC/3270 or PC400 can perform the DDE_UNADVISE, PC/3270 or PC400 returns an ACK message containing positive status information to the client:

```
WM_DDE_ACK(wStatus, aMOUSE)
```

If PC/3270 or PC400 cannot perform the DDE_UNADVISE, PC/3270 or PC400 returns an ACK message containing negative status information and the following return codes in the low-order byte of the wStatus word.

Return Code	Explanation
1	Advise has not started yet.
9	A system error occurred.

Stop Read SF

PC/3270		*PC400*
DOS	Windows	
–	X	–

The Stop Read SF function ends a client application's ability to read structured field data.

The client sends the following command to perform the Stop Read SF function.

```
PostMessage( hServerWnd,
             WM_DDE_UNADVISE,
             hClientWnd,
              MAKELONG(NULL, aSF) );
```

where:

aSF Identified structured field as the item.

PC/3270 response

If PC/3270 can perform the DDE_UNADVISE, PC/3270 returns an ACK message containing positive status information to the client:

```
WM_DDE_ACK(wStatus, aSF)
```

If PC/3270 cannot perform the DDE_UNADVISE, PC/3270 returns an ACK message containing negative status information and the following return codes in the low-order byte of the wStatus word.

Return Code	Explanation
1	Advise has not started yet.
9	A system error occurred.

Stop Session Advise

PC/3270		*PC400*
DOS	Windows	
–	X	X

The Stop Session Advise function disconnects a link between PC/3270 or PC400 and the client. The client sends the following command to perform the Stop Session Advise function.

```
PostMessage( hServerWnd,
             WM_DDE_UNADVISE,
             hClientWnd,
              MAKELONG(NULL, aItem) );
```

where:

aItem Specifies the item of information being requested; in this case, the value can be PS, OIA, TRIMRECT, or NULL.

If the value of *aItem* is NULL, then the client has requested termination of all active notifications for the conversation.

PC/3270 or PC400 Response

If PC/3270 or PC400 can perform the DDE_UNADVISE, PC/3270 or PC400 returns an ACK message containing positive status information to the client:

```
WM_DDE_ACK(wStatus, aItem)
```

If PC/3270 or PC400 cannot perform the DDE_UNADVISE, PC/3270 or PC400 returns an ACK message containing negative status information and the following return codes in the low-order byte of the wStatus word.

Return Code	Explanation
1	Advise has not started yet.
9	A system error occurred.

Terminate Session Conversation

PC/3270		*PC400*
DOS	Windows	
–	X	X

The Terminate Session Conversation function disconnects the client from the PC/3270 or PC400 session the client has previously started a conversation with.

The client sends the following command to terminate a session conversation.

```
SendMessage( hServerWnd,
             WM_DDE_TERMINATE,
             hClientWnd,
              MAKELONG(NULL, NULL) );
```

PC/3270 or PC400 Response

PC/3270 or PC400 acknowledges the terminate command with a terminate message:

WM_DDE_TERMINATE

Terminate Structured Field Conversation

PC/3270		*PC400*
DOS	Windows	
–	X	–

The Terminate Structured Field Conversation function disconnects the client from a structured field conversation.

The client sends the following command to terminate a structured field conversation.

```
SendMessage( hServerWnd,
             WM_DDE_TERMINATE,
             hClientWnd,
              MAKELONG(NULL, NULL) );
```

PC/3270 response

PC/3270 acknowledges the terminate command with a terminate message:

WM_DDE_TERMINATE

Terminate System Conversation

PC/3270		*PC400*
DOS	Windows	
–	X	X

This disconnects the client from a system conversation.

The client sends the following command to terminate a system conversation.

```
SendMessage( hServerWnd,
             WM_DDE_TERMINATE,
             hClientWnd,
              MAKELONG(NULL, NULL) );
```

PC/3270 or PC400 Response

PC/3270 or PC400 acknowledges the terminate command with this message:

WM_DDE_TERMINATE

When the user closes a PC/3270 or PC400 session, any global memory blocks that were allocated by PC/3270 or PC400 will be freed by Windows. This can cause problems for the client if the client retains any of these global memory objects for long periods of time. If the client application needs to keep the information in a global memory item for a long period of time, it is suggested that the client make a copy of global memory item into a global memory item the client application owns.

Write SF

PC/3270		PC400
DOS	Windows	
–	X	–

The Write SF function allows a client application to write structured field data to the host application.

Note: The client must call the Set Structured Field Service Condition function before invoking the Write SF function.

The client sends the following message to write structured field data.

```
PostMessage( hServerWnd,
             WM_DDE_POKE,
             hClientWnd,
              MAKELONG(hData, aSF) );
```

where:

hData Identifies a handle to a Windows global memory object containing:

```
typedef struct tagWRITESF
{
 unsigned     unused:12,              //
 unsigned     fRelease:1,             //
 unsigned     fReserved:3;            //
 int          cfFormat;               // Always CF_DSPTEXT
 WORD           uSFLength;                // Length of SF data
 char         Work[8];                // Work area
 char         szSFData[1];            // SF data
} WRITESF, FAR  lpWRITESF;
```

aSF Identified structured field as the item.

PC/3270 response

PC/3270 receives structured field data and sends them to the host application. If the data transmission completes successfully, then PC/3270 returns an ACK message:

```
WM_DDE_ACK(wStatus, aSF)
```

Otherwise PC/3270 returns an negative ACK message containing the following return codes in the low-order byte of wStatus:

Return Code	Explanation
2	Length is not valid.
6	The specified format is not valid.
9	A system error occurred.

Summary of DDE Functions

The following table lists the DDE functions that can be used with PC/3270 or PC400. The table lists the name of the DDE function, the command the client sends to PC/3270 or PC400, and the values that can be used for the variables in the client command.

Table 18 (Page 1 of 4). DDE Function Summary

Function name	Client command
Code Conversion	PostMessage(hServerWnd, WM_DDE_POKE, hClientWnd, MAKELONG(hData, aCONV));
	hData = Handle to a global memory object
Find Field	PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELONG(cfFormat, aFIELD));
	cfFormat = CF_DSPTEXT
Get Keystrokes	PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELONG(cfFormat, aKEYS));
	cfFormat = CF_DSPTEXT
Get Mouse Input	PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELONG(cfFormat, aMOUSE));
	cfFormat = CF_TEXT | CF_DSPTEXT
Get Number of Close Requests	PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELONG(cfFormat, aCLOSE));
	cfFormat = CF_DSPTEXT
Get Operator Information Area	PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELONG(cfFormat, aOIA));
	cfFormat = CF_DSPTEXT
Get Partial Presentation Space	PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELONG(cfFormat, aEPS));
	cfFormat = CF_TEXT | CF_DSPTEXT
Get Presentation Space	PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELONG(cfFormat, aPS));
	cfFormat = CF_TEXT | CF_DSPTEXT
Get Session Status	PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELONG(cfFormat, aSSTAT));
	cfFormat = CF_TEXT
Get System Configuration	PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELONG(cfFormat, aSYSCON));
	cfFormat = CF_TEXT

Table 18 (Page 2 of 4). DDE Function Summary

Function name	Client command
Get System Formats	PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELONG(cfFormat, aFORMATS));
	cfFormat = CF_TEXT
Get System Status	PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELONG(cfFormat, aSTATUS));
	cfFormat = CF_TEXT
Get System SysItems	PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELONG(cfFormat, aSYSITEMS));
	cfFormat = CF_TEXT
Get System Topics	PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELONG(cfFormat, aTOPICS));
	cfFormat = CF_TEXT
Get Trim Rectangle	PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELONG(cfFormat, aTRIMRECT));
	cfFormat = CF_TEXT
Initiate Session Conversation	SendMessage(-1, WM_DDE_INITIATE, hClientWnd, MAKELONG(aIBM327 , aSessionN));
	N = a session letter A through Z.
Initiate Structured Field Conversation	SendMessage(-1, WM_DDE_INITIATE, hClientWnd, MAKELONG(aIBM327 , aLUN_xxxx)
	N = a session letter A through Z. xxxx = a user defined string.
Initiate System Conversation	SendMessage(-1, WM_DDE_INITIATE, hClientWnd, MAKELONG(aIBM327 , aSystem));
Put Data to Presentation Space	PostMessage(hServerWnd, WM_DDE_POKE, hClientWnd, MAKELONG(hData, aEPS));
	hData = Handle to a global memory object
Search for String	PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELONG(cfFormat, aSTRING));
	cfFormat = CF_DSPTEXT
Send Keystrokes	PostMessage(hServerWnd, WM_DDE_POKE, hClientWnd, MAKELONG(hData, aKEYS));
	hData = Handle to a global memory object

Table 18 (Page 3 of 4). DDE Function Summary

Function name	Client command
Session Execute Macro	PostMessage(hServerWnd, WM_DDE_EXECUTE, hClientWnd, MAKELONG(NULL, hCommands));
	hCommands = Handle to a global memory object
Set Cursor Position	PostMessage(hServerWnd, WM_DDE_POKE, hClientWnd, MAKELONG(hData, aSETCURSOR));
	hData = Handle to a global memory object
Set Mouse Intercept Condition	PostMessage(hServerWnd, WM_DDE_POKE, hClientWnd, MAKELONG(hData, aMOUSE));
	hData = Handle to a global memory object
Set Presentation Space Service Condition	PostMessage(hServerWnd, WM_DDE_POKE, hClientWnd, MAKELONG(hData, aEPSCOND));
	hData = Handle to a global memory object
Set Session Advise Condition	PostMessage(hServerWnd, WM_DDE_POKE, hClientWnd, MAKELONG(hData, aPSCOND));
	hData = Handle to a global memory object
Set Structured Field Service Condition	PostMessage(hServerWnd, WM_DDE_POKE, hClientWnd, MAKELONG(hData, aSFCOND));
	hData = Handle to a global memory object
Start Close Intercept	SendMessage(hServerWnd, WM_DDE_ADVISE, hClientWnd, MAKELONG(hOptions, aCLOSE));
	hOptions = Handle to a global memory object
Start Keystroke Intercept	SendMessage(hServerWnd, WM_DDE_ADVISE, hClientWnd, MAKELONG(hOptions, aKEYS));
	hOptions = Handle to a global memory object
Start Mouse Input Intercept	PostMessage(hServerWnd, WM_DDE_ADVISE, hClientWnd, MAKELONG(hOptions, aMOUSE));
	hOptions = Handle to a global memory object
Start Read SF	PostMessage(hServerWnd, WM_DDE_ADVISE, hClientWnd, MAKELONG(hOptions, aSF));
	hOptions = Handle to a global memory object
Start Session Advise	PostMessage(hServerWnd, WM_DDE_ADVISE, hClientWnd, MAKELONG(hOptions, aItem));
	hOptions = Handle to a global memory object aItem = OIA | PS | TRIMRECT

Table 18 (Page 4 of 4). DDE Function Summary

Function name	Client command
Stop Close Intercept	PostMessage(hServerWnd, WM_DDE_UNADVISE, hClientWnd, MAKELONG(NULL, aCLOSE));
Stop Keystroke Intercept	PostMessage(hServerWnd, WM_DDE_UNADVISE, hClientWnd, MAKELONG(NULL, aKEYS));
Stop Mouse Input Intercept	PostMessage(hServerWnd, WM_DDE_UNADVISE, hClientWnd, MAKELONG(NULL, aMOUSE));
Stop Read SF	PostMessage(hServerWnd, WM_DDE_UNADVISE, hClientWnd, MAKELONG(NULL, aSF));
Stop Session Advise	PostMessage(hServerWnd, WM_DDE_UNADVISE, hClientWnd, MAKELONG(NULL, aItem));
	aItem = OIA | PS | TRIMRECT | NULL
Terminate Session Conversation	SendMessage(hServerWnd, WM_DDE_TERMINATE, hClientWnd, MAKELONG(NULL, NULL));
Terminate Structured Field Conversation	SendMessage(hServerWnd, WM_DDE_TERMINATE, hClientWnd, MAKELONG(NULL, NULL));
Terminate System Conversation	SendMessage(hServerWnd, WM_DDE_TERMINATE, hClientWnd, MAKELONG(NULL, NULL));
Write SF	PostMessage(hServerWnd, WM_DDE_POKE, hClientWnd, MAKELONG(hData, aSF));
	hData = Handle to a global memory object

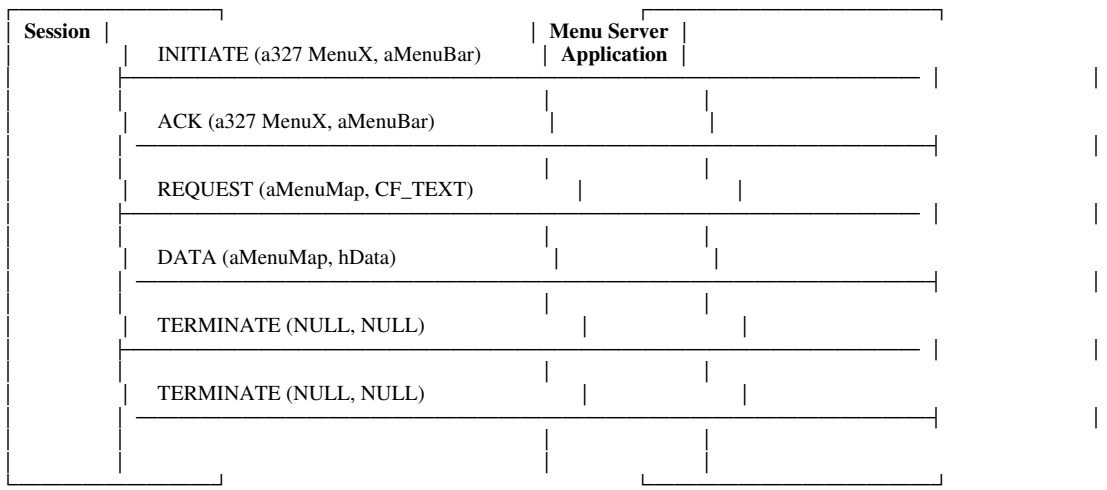
DDE Menu Item API

PC/3270 or PC400 supports the addition, deletion, and changing of attributes of a dynamic menu item to the session menu bar. A menu will then be created for this menu item with space for up to 16 submenu items.

PC/3270 or PC400 supports two kinds of DDE conversation. One is PC/3270 or PC400, which acts as a DDE menu client application, and the other is PC/3270 or PC400, which acts as a DDE menu server.

DDE Menu Client

To add, delete, and change menu items, the following DDE conversation must take place between the session and DDE menu server application.



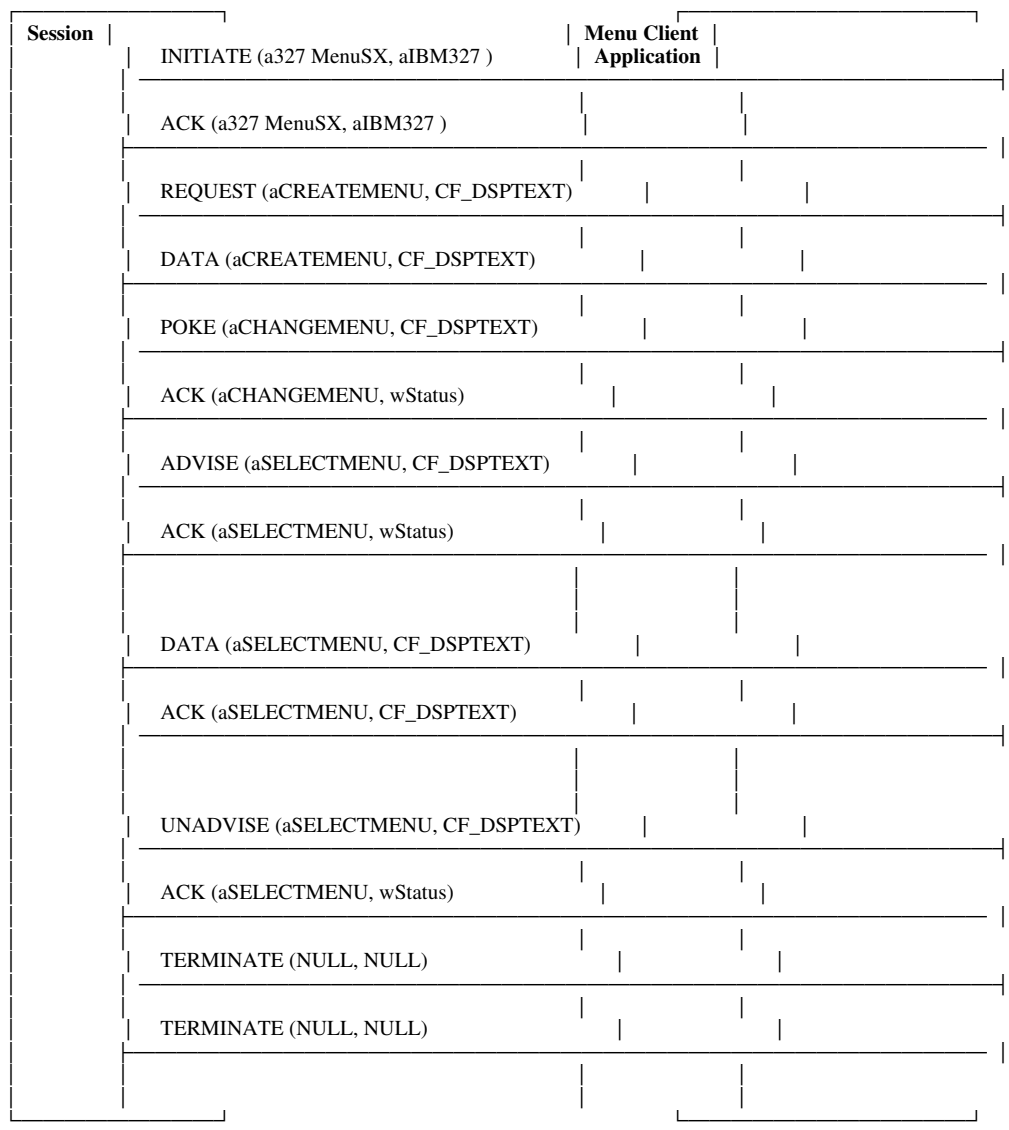
The following data hierarchy details the menu map PC/3270 or PC400 expects when adding a dynamic menu item and submenu to a session menu bar:

```
POPUP    "MyMenu"
BEGIN
    MENUITEM "Send Files to Host",              SEND
    MENUITEM "Receive Files from Host",       RECEIVE
   MENUITEM  SEPARATOR
    MENUITEM "Convert Files",                    CONVERT
END
```

When the user selects a menu item from the new menu, PC/3270 or PC400 will send a DDE Initiate with 3270MenuN or 5250MenuN as the application and itemN token as the topic. If an ACK is received from the DDE application, PC/3270 or PC400 will inhibit the session from accepting user input. The menu client application can then display a dialog, and so on. When the menu server application has completed processing of the menu item, it will send a DDE Terminate to signal PC/3270 or PC400 the process is complete. PC/3270 or PC400 will then reenable the window for the user.

DDE Menu Server

To add, delete, and change menu items, the following DDE conversation must take place between the session and a DDE menu client application.



When the user selects a menu item from the new menu, PC/3270 or PC400 will send a DDE DATA with aSELECTMENU as the item. When PC/3270 or PC400 sends DDE DATA to the client application, PC/3270 or PC400 will inhibit the session from accepting user input. The menu client application can then display a dialog, and so on. When the menu client application has completed processing of the menu item, it will send a DDE ACK to signal PC/3270 or PC400 the process is complete. PC/3270 or PC400 will then reenable the window for the user.

DDE Menu Functions

The following table lists the DDE Menu Item API functions that are available for use with PC/3270 or PC400 and the page in this section, where each function is more fully documented. PC/3270 Windows mode and PC400 provide all of the following functions.

Function	Page
Change Menu Item	267
Create Menu Item	273
Initiate Menu Conversation	275
Start Menu Advise	276
Stop Menu Advise	278
Terminate Menu Conversation	279

Refer to “Summary of DDE Menu Item API Functions” on page 280 for a summary of the DDE Menu Item API functions.

Change Menu Item

PC/3270		PC400
DOS	Windows	
–	X	X

The Change Menu Item function appends, deletes, inserts, modifies, and removes menu items. The client sends the following message to the session to change a menu.

```
PostMessage( hServerWnd,
             WM_DDE_POKE,
             hClientWnd,
              MAKELONG(hData, aCHANGEMENU) );
```

where:

hData

Identifies a handle to a Windows global memory object that contains the requests for changing a menu. The global memory object contains the following structure:

```
typedef struct tagChangeMenu
{
 unsigned unused:13;         //    unused
 unsigned fRelease:1;       // Session frees memory
 unsigned freserved:2;     //    reserved
  int          cfFormat;          //  Always  CF_DSPTEXT
 HANDLE     hMenu;                 // Handle of the menu item
  WORD        wPosition;        //  The  position  of  the  menu
                                 //  item
 WORD         wIDNew;               // The menu ID of the new
                                  // menu item
 WORD         wOperation;       // Specifies the operation
 WORD         wFlags;             // Specifies the options
 unsigned char szItemName[1]; // String of the item
} CHANGEMENU, FAR  lpCHANGEMENU;
```

The following operations are supported:

```
MF_APPEND    // Appends a menu item to the end of a menu.
MF_CHANGE    // Modifies a menu item in a menu.
MF_DELETE    // Deletes a menu item from a menu, destroying
                // the menu item.
MF_INSERT    // Inserts a menu item into a menu.
MF_REMOVE    // Removes a menu item from a menu but does not
                // destroy the menu item.
```

If the MF_APPEND is specified in the wOperation field, the following fields must be filled:

hMenu

Identifies the menu to be appended. To append a new item to a pop-up menu, specify the handle that is returned from PC/3270 or PC400 when Create Menu Item function is executed. To append a new item to a top-level menu bar, specify NULL.

wIDNew

Specifies the command ID of the new menu item. If a new item is added to the top-level menu bar, the handle of the menu item returned from PC/3270 or PC400 when Create Menu Item function is executed.

wFlags

The following options can be set:

```
MF_CHECKED          // Places a checkmark next to
                    // the item.
MF_DISABLED         // Disables the menu item so
                    // that it cannot be selected,
                    // but does not gray it.
MF_ENABLED          // Enables the menu item so that
                    // it can be selected and
                    // restores from its grayed
                    // state.
MF_GRAYED           // Disables the menu item so
                    // that it cannot be selected,
                    // and grays it.
MF_MENUBARBREAK     // Same as MF_MENUBREAK except
                    // that for pop-up menus,
                    // separates the new column from
                    // the old column with a
                    // vertical line.
MF_MENUBREAK        // Places the item on a new line
                    // for menu bar items.
                    // For pop-up menus, places the
                    // item in a new column, with
                    // no dividing line between the
                    // columns.
MF_SEPARATOR        // Draws a horizontal dividing
                    // line.  Can only be used in a
                    // pop-up menu.  This line cannot
                    // be grayed, disabled, or
                    // highlighted.  The wIDNew and
                    // szItemName fields are
                    // ignored.
MF_UNCHECKED        // Does not place a checkmark
                    // next to the item (default).
```

szItemName

Specifies the content of the new menu item. Contains a null-terminated character string.

If the MF_CHANGE is specified in the wOperation field, fill these fields:

hMenu

Identifies the menu to be changed. To change an item of a pop-up menu, specify the handle that is returned from PC/3270 or PC400 when Create Menu Item function is executed. To change an item to a top-level menu bar, specify NULL.

nPosition

Specifies the menu item to be changed. The interpretation of the wPosition parameter depends on the setting of the wFlags parameter.

MF_BYPOSITION

Specifies the position of the existing menu item. The first item in the menu is at position zero.

MF_BYCOMMAND

Specifies the command ID of the existing menu item.

wIDNew

Specifies the command ID of the menu item. If an item of the top-level menu bar is changed, the handle of the menu item returned from PC/3270 or PC400 when Create Menu Item function is executed.

wFlags

The following options can be set:

```
MF_BYCOMMAND        // Specifies that the nPosition
                    // parameter gives the menu
                    // item control ID number.
                    // This is the default if
                    // neither MF_BYCOMMAND nor
                    // MF_BYPOSITION is set.
MF_BYPOSITION       // Specifies that the nPosition
                    // parameter gives the position
                    // of the menu item to be
                    // changed rather than an ID
                    //  number.
MF_CHECKED          // Places a checkmark next to
                    // the item.
MF_DISABLED         // Disables the menu item so
                    // that it cannot be selected,
                    // but does not gray it.
MF_ENABLED          // Enables the menu item so
                    // that it can be selected and
                    // restores from its grayed
                    //  state.
MF_GRAYED           // Disables the menu item so
                    // that it cannot be selected,
                    // and grays it.
MF_MENUBARBREAK   // Same as MF_MENUBREAK except
                    // that for pop-up menus,
                    // separates the new column
                    // from the old column with a
                    // vertical line.
MF_MENUBREAK        // Places the item on a new
                    // line for menu bar items.
                    // For pop-up menus, places the
                    // item in a new column, with
                    // no dividing line between
                    // the columns.
MF_SEPARATOR        // Draws a horizontal dividing
                    // line.   Can only be used in
                    // a pop-up menu.   This line
                    // cannot be grayed, disabled,
                    // or highlighted.   The wIDNew
                    // and szItemName fields are
                    //  ignored.
MF_UNCHECKED        // Does not place a checkmark
                    // next to the item (default).
```

szItemName

Specifies the content of the menu item. Contains a null-terminated character string.

If the MF_DELETE is specified in the wOperation field, fill these fields:

hMenu

Identifies the menu to be deleted. To delete an item from a pop-up menu, specify the handle that is returned from PC/3270 or PC400 when Create Menu Item function is executed. To delete an item from a top-level menu bar, specify NULL.

nPosition

Specifies the menu item to be deleted. The interpretation of the nPosition parameter depends on the setting of the wFlags parameter.

MF_BYPOSITION
Specifies the position of the existing menu item. The first item in the menu is at position zero.

MF_BYCOMMAND
Specifies the command ID of the existing menu item.

wFlags

The following options can be set:

```
MF_BYCOMMAND        // Specifies that the nPosition
                    // parameter gives the menu
                    // item control ID number.
                    // This is the default if
                    // neither MF_BYCOMMAND nor
                    // MF_BYPOSITION is set.
MF_BYPOSITION       // Specifies that the nPosition
                    // parameter gives the position
                    // of the menu item to be
                    // deleted rather than an ID
                    //  number.
```

If the MF_INSERT is specified in the wOperation field, the following fields must be filled:

hMenu

Identifies the menu to be inserted. To insert an item to a pop-up menu, specify the handle that is returned from PC/3270 or PC400 when Create Menu Item function is executed. To change an item to a top-level menu bar, specify NULL.

nPosition

Specifies the menu item before the new menu item is to be inserted. The interpretation of the nPosition parameter depends on the setting of the wFlags parameter.

MF_BYPOSITION
Specifies the position of the existing menu item. The first item in the menu is at position zero.

MF_BYCOMMAND
Specifies the command ID of the existing menu item.

wIDNew

Specifies the command ID of the menu item or, if an item of the top-level menu bar is changed, the handle of the menu item returned from PC/3270 or PC400 when Create Menu Item function is executed.

wFlags

The following options can be set:

```
MF_BYCOMMAND        // Specifies that the nPosition
                    // parameter gives the menu
                    // item control ID number. This
                    // is the default if neither
                    // MF_BYCOMMAND nor MF_BYPOSITION
                    // is set.
MF_BYPOSITION       // Specifies that the nPosition
                    // parameter gives the position
                    // of the menu item to be
                    // changed rather than an ID
                    //  number.
MF_CHECKED          // Places a checkmark next to
                    // the item.
MF_DISABLED         // Disables the menu item so
                    // that it cannot be selected,
                    // but does not gray it.
MF_ENABLED          // Enables the menu item so
                    // that it can be selected and
                    // restores from its grayed
                    //  state.
MF_GRAYED           // Disables the menu item so
                    // that it cannot be selected,
                    // and grays it.
MF_MENUBARBREAK     // Same as MF_MENUBREAK except
                    // that for pop-up menus,
                    // separates the new column
                    // from the old column with a
                    // vertical line.
MF_MENUBREAK        // Places the item on a new
                    // line for menu bar items.
                    // For pop-up menus, places the
                    // item in a new column, with
                    // no dividing line between the
                    //  columns.
MF_SEPARATOR        // Draws a horizontal dividing
                    // line.    Can only be used in
                    // a pop-up menu.    This line
                    // cannot be grayed, disabled,
                    // or highlighted. The wIDNew
                    // and szItemName fields are
                    //  ignored.
MF_UNCHECKED        // Does not place a checkmark
                    // next to the item (default).
```

szItemName

Specifies the content of the menu item. Contains a null-terminated character string.

If the MF_REMOVE is specified in the wOperation field, the following fields must be filled:

hMenu

Identifies the menu to be removed. To remove an item from a pop-up menu, specify the handle that is returned from PC/3270 or PC400 when Create Menu Item function is executed. To remove an item from a top-level menu bar, specify NULL.

nPosition

Specifies the menu item to be removed. The interpretation of the nPosition parameter depends upon the setting of the wFlags parameter.

MF_BYPOSITION

Specifies the position of the existing menu item. The first item in the menu is at position zero.

MF_BYCOMMAND

Specifies the command ID of the existing menu item.

wFlags

The following options can be set:

```
MF_BYCOMMAND        // Specifies that the nPosition
                    // parameter gives the menu
                    // item control ID number.
                    // This is the default if
                    // neither MF_BYCOMMAND nor
                    // MF_BYPOSITION is set.
MF_BYPOSITION       // Specifies that the nPosition
                    // parameter gives the
                    // position of the menu item to
                    // be removed rather than an ID
                    //  number.
```

PC/3270 or PC400 Response

PC/3270 or PC400 receives the requests to change a menu and processes them. If the requests cannot be accepted, PC/3270 or PC400 returns a negative ACK message containing the following status codes in the low-order byte of the wStatus word. Otherwise, PC/3270 or PC400 returns a positive ack message signalling that the keystrokes have been sent.

WM_DDE_ACK(wStatus, aCHANGEMENU)

Return code	Explanation
1	The specified parameters are not valid.
6	The specified format is not valid.
9	A system error occurred.

Create Menu Item

PC/3270		*PC400*
DOS	Windows	
–	x	x

The Create Menu Item function requests PC/3270 or PC400 to add a menu item to the menu bar. A pop-up menu will be created at the same time, but it is initially empty and can be filled with menu items by using this function. The string of the new menu item that will be added to a top-level menu bar, is also specified by using the change menu item function.

The client sends the following message to create a menu item.

```
PostMessage( hServerWnd,
             WM_DDE_REQUEST,
             hClientWnd,
              MAKELONG(cfFormat, aCREATEMENU) );
```

where:

cfFormat

Identifies the format for the ID of the new menu item. The valid value is CF_DSPTEXT.

aCREATEMENU

Identifies the create menu item.

PC/3270 or PC400 response

PC/3270 or PC400 returns the handle of the newly created menu item in a dde data message if the PC/3270 or PC400 can create a menu item.

WM_DDE_DATA(hData, aCREATEMENU)

where:

hData

Identifies a handle to a windows global memory object that contains the handle of the menu item. The global memory object contains the following structure:

```
typedef struct tagcreatemenu
}
 unsigned     unused:12,            //     unused
 unsigned     fresponse:1,         // true = dd_request response
 unsigned     frelease:1,          // true = client releases memory
 unsigned     reserved:1,           //     reserved
 unsigned     fackreq:1,            // true = dde_ack is required
 int          cfformat;                      // always cf_dsptext
 handle       hmemuitem;                     // handle of the menu item
} CREATEMENU, FAR  lpCREATEMENU;
```

or

WM_DDE_ACK(wStatus, aCREATEMENU)

If PC/3270 or PC400 cannot create a menu item, the following status codes are returned in the low-order byte of the wStatus word:

Return Code	Explanation
6	The specified format is not valid.
9	A system error occurred.

Initiate Menu Conversation

PC/3270		*PC400*
DOS	Windows	
–	X	X

The Initiate Menu Conversation function connects a client application to an available session of PC/3270 or PC400. Once a menu conversation is established, the session menu is reserved exclusively for the client until the conversation is terminated.

The client application sends the following message to initiate a DDE conversation with a menu:

```
SendMessage( -1,
             WM_DDE_INITIATE,
             hClientWnd,
              MAKELONG(aIBM327 , a327 MenuSN) );
```

where:

aIBM3270

Identifies the application atom. The string used to create atom aIBM3270 is IBM3270 In the PC400, the application atom is aIBM5250 and the string IBM5250 is used to create it.

a3270MenuSN

Identifies the topic atom. The string used to create atom a3270MenuSN is 3270MenuS appended with the session ID A, B, ..., Z. In the PC400, the topic atom is a5250MenuSN and the string 5250MenuS appended with the session ID A, B, ..., Z. is used to create it.

PC/3270 or PC400 Response

If PC/3270 or PC400 can support a conversation with the client application, PC/3270 or PC400 acknowledges the INITIATE transaction with:

```
WM_DDE_ACK(aIBM327 , a327 MenuSN)
```

Start Menu Advise

PC/3270		PC400
DOS	Windows	
–	X	X

The Start Menu Advise function allows a client application to process a user defined routine when the menu item that is added by the client application, is selected. After using this function, the client receives DATA messages indicating which menu item is selected.

The client sends the following command to begin a menu advise.

```
PostMessage( hServerWnd,
             WM_DDE_ADVISE,
             hClientWnd,
              MAKELONG(hOptions, aSELECTMENU) );
```

where:

hOptions

Is a handle to a Windows global memory object, with the following structure:

```
typedef struct tagOPTIONS
{
 unsigned reserved:14;          // Reserved
 unsigned fDeferUpd:1;           // Send notification only
                                     // (Must be  )
 unsigned fAckReq:1;              // Client will ACK all notices
                                     // (Must be 1)
 WORD        cfFormat;              // Always CF_DSPTEXT
}    OPTIONS, FAR  lpOPTIONS;
```

aSELECTMENU

Identified a menu advise as the item.

PC/3270 or PC400 Response

PC/3270 or PC400 receives the Start Menu Advise and returns an ACK message if it can start the function. Otherwise, a negative ACK message will be returned to the client with the following return codes in the low-order byte of the wStatus field.

Return Code	Explanation
1	Menu Advise has been already started.
6	The specified format is not valid.
9	A system error occurred.

```
WM_DDE_ACK(wStatus, aSELECTMENU)
```

Once the menu item (added to the client application) is selected, the client receives DATA messages notifying it which menu item is selected:

```
WM_DDE_DATA(hData, aSELECTMENU)
```

where:

hData

Identifies a handle to a Windows global memory object containing:

```
typedef struct tagSELECTMENU
{
 unsigned     Unused:12,          //     unused
 unsigned     fResponse:1,       // TRUE = DD_REQUEST response
 unsigned     fRelease:1,        // TRUE = Client releases memory
 unsigned     reserved:1,        //     reserved
 unsigned     fAckReq:1,          // TRUE = DDE_ACK is required
 int          cfFormat;                  // Always CF_DSPTEXT
 WORD         uIDSelected;                // Command ID of the
                                          // selected menu item
} SELECTMENU, FAR  lpSELECTMENU;
```

The DATA messages continue until a Stop Menu Advise message is sent to PC/3270 or PC400.

Stop Menu Advise

PC/3270		*PC400*
DOS	Windows	
–	X	X

The Stop Menu Advise function ends a client application's ability to process a user-defined routine when the menu item added by the client application is selected. The client sends the following command to perform the Stop Menu Advise function.

```
PostMessage( hServerWnd,
             WM_DDE_UNADVISE,
             hClientWnd,
              MAKELONG(NULL, aSELECTMENU) );
```

where:

aSELECTMENU

Identifies a menu advise as the item.

PC/3270 or PC400 Response

If PC/3270 or PC400 can perform the DDE_UNADVISE, PC/3270 or PC400 returns an ACK message containing positive status information to the client:

```
WM_DDE_ACK(wStatus, aCLOSE)
```

If PC/3270 or PC400 cannot perform the DDE_UNADVISE, PC/3270 or PC400 returns an ACK message containing negative status information and the following return codes in the low-order byte of the wStatus word:

Return Code	Explanation
1	Advise has not started yet.
9	A system error occurred.

Terminate Menu Conversation

PC/3270		*PC400*
DOS	Windows	
–	X	X

The Terminate Menu Conversation function disconnects the client from the PC/3270 or PC400 session with which a conversation had been previously started.

The client sends the following command to terminate a session conversation:

```
SendMessage( hServerWnd,
             WM_DDE_TERMINATE,
             hClientWnd,
              MAKELONG(NULL, NULL) );
```

PC/3270 or PC400 Response

PC/3270 or PC400 acknowledges the terminate command with this message:

WM_DDE_TERMINATE

Summary of DDE Menu Item API Functions

The following table lists the DDE Menu Item API functions that can be used with PC/3270 or PC400. The table lists the name of the DDE Menu Item API function, the command the client sends to PC/3270 or PC400, and the values that can be used for the variables in the client command.

Table 19. DDE Menu Item API Function Summary

Function name	Client command
Change Menu Item	PostMessage(hServerWnd, WM_DDE_POKE, hClientWnd, MAKELONG(hData, aCHANGEMENU));
	hData = Handle to a global memory object
Create Menu Item	PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELONG(cfFormat, aCREATEMENU));
	cfFormat = CF_DSPTEXT
Initiate Menu Conversation	SendMessage(-1, WM_DDE_INITIATE, hClientWnd, MAKELONG(aIBM327 , a327 MenuSN));
	N = a session letter A through Z
Start Menu Advise	PostMessage(hServerWnd, WM_DDE_ADVISE, hClientWnd, MAKELONG(hOption, aSELECTMENU));
	hData = Handle to a global memory object
Stop Menu Advise	PostMessage(hServerWnd, WM_DDE_UNADVISE, hClientWnd, MAKELONG(NULL, aSELECTMENU));
Terminate Menu Conversation	SendMessage(hServerWnd, WM_DDE_TERMINATE, hClientWnd, MAKELONG(NULL, NULL));

Chapter 9. Using DDE Functions with DDE Client Application (Visual Basic)

Windows allows users to run multiple Windows application programs and to exchange data among Windows application programs. DDE (Dynamic Data Exchange) is a method that allows users this data exchange. Data exchange among Windows application programs can be considered as conversations between *Server* and *Client* application programs. The client application is an application program that starts DDE and the server application is an application program that respond to the client application.

The client application needs three names (application program name, topic name, item name) that are recognized by the server application to start the conversation to exchange data. The client application starts DDE conversation with the server application by specifying application program and topic names, and defines the exchange data by specifying item name.

PC/3270 or PC400 has a function as a DDE server, and can establish DDE conversations with other Windows application programs (Microsoft Visual Basic, Microsoft Excel, Microsoft Word) that have DDE client functions.

PC/3270 Windows mode provides new user interface formats of the DDE functions. The new formats are similar to Visual Basic interface. The following functions support the new formats as well as the old formats:

Find Field
Get Partial Presentation Space
Get Trim Rectangle
Put Data to Presentation Space
Search for String
Start Session Advise

Using PC/3270 and PC400 DDE Interface

To start DDE conversation and data exchange with PC/3270 or PC400, client application programs need to know the application program name, topic name, and item name that PC/3270 or PC400 can recognize, The exchange data type between an application program and PC/3270 or PC400 is defined with the combination of these names.

Table 20. Naming Scheme for Data Items

Level	Description	Example
Application	Windows task or a specific task of the application. In this book, application programs are PC/3270 or PC400.	IBM3270
Topic	A specific part of application programs	SessionA
Item	Type of data passed during DDE conversation	PS (Presentation Space)

Application

As a Windows DDE server, PC/3270 or PC400 supports application name *IBM3270* or *IBM5250*.

© Copyright IBM Corp. 1996

Topic Topic specifies the corresponding topic in the application. The following table shows the topics available to the users:

Table 21. Topics for PC/3270 or PC400

Topic	Conversation Name	Conversation Type
System	System conversation	Cold link
SessionA, SessionB, ..., SessionZ	Session conversation	Cold link and Hot link
SessA_xxxx, SessB_xxxx, ..., SessZ_xxxx	Session conversation	Hot link

Item The client application programs can exchange data and information with PC/3270 or PC400. Types of data and information are specified by Item name.

Explanations for topic conversation procedures and data item to be used follows.

System Conversation

To use the PC/3270 or PC400 system DDE interface, do as follows:

1. Start system conversation
2. Demand system information
3. Terminate system conversation

Starting DDE Conversation with PC/3270 or PC400 System

To use the DDE interface with the PC/3270 or PC400 system, the client application should start a DDE conversation with PC/3270 or PC400 first. To start DDE conversation, specify *IBM3270* or *IBM5250* as an application name and *System* as a topic name in the DDE function (Initiate) in the client application.

Requesting System Information

After starting DDE conversation, the client application can request data or information using the DDE function. System information can be requested by specifying the following item names in the DDE function (Request) in the client application:

Item	Return data/information	DDE function
Formats	List of supported Windows clipboard format	Get System Formats
Status	Each session status information	Get System Status
SysCon	Information of emulator support level and other system related values	Get System Configuration
SysItems	List of available data items	Get System SysItems
Topics	List of available topics	Get System Topics

Terminating DDE Conversation with PC/3270 or PC400 System

To complete the conversation, the client application needs to terminate the DDE conversation with PC/3270 or PC400. To terminate the conversation, use the DDE function (Terminate) in the client application.

Session Conversation

To use the PC/3270 or PC400 session DDE interface, do as follows:

1. Start session conversation
2. Use DDE functions (Request, Poke, Execute)
3. Terminate session conversation

Starting DDE Conversation with the Session

To use the DDE interface with PC/3270 or PC400 sessions, the client application should start DDE conversation with PC/3270 or PC400. To start DDE conversation, specify *IBM3270* or *IBM5250* as an application name and *SessionA*, *SessionB*, ..., *SessionZ* as topic names in the DDE function (Initiate) in the client application.

Requesting Data or Information

After starting DDE conversation, the client application can request data or information using the DDE function. System information can be requested by specifying the following item names in the DDE function (Request) in the client application:

Item	Return data/information	DDE function
EPS(pos,len,bEOF)	All or a part of session presentation space	Get Partial Presentation Space
FIELD(pos,"type")	Field information	Find Field
OIA	Operator Information Area (OIA) status line information	Get Operator Information Area
PS	Session presentation space	Get Presentation Space
SSTAT	Session status information	Get System Status
STRING(pos,bDir,"string")	String offset start	Search for string
TRIMRECT *1	Session presentation space of trim rectangle	Get Trim Rectangle

Note:

***1** Parameter can be added

Sending Data to Emulator Window (Poke)

After starting DDE conversation, the client application can send data to PC/3270 or PC400 sessions using the DDE functions. The following chart shows the valid items for the DDE functions.

Item	Explanation	DDE Function
EPS(pos,bEOF)	Sends ASCII data string to the host presentation space	Put Data to Presentation Space
SETCURSOR	Sets the cursor position	Set Cursor Position

Executing Commands

After starting DDE conversation, the client application can send commands to PC/3270 or PC400 session window using the DDE functions. Specify the command in the DDE function (Execute) of the client application. Refer to “Session Execute Macro” on page 223 for details.

Terminating Session DDE Conversation

The client application should terminate the DDE conversation with PC/3270 or PC400 when completing the task. To terminate the conversation, use the DDE function (Terminate) in the client application.

Session Conversation (Hot Link)

To use the PC/3270 or PC400 session DDE interface, do as follows:

1. Start session conversation
2. Start the advice function
3. Stop the advice function
4. Terminate session conversation

Starting DDE Conversation with the Session (Hot Link)

To use DDE interface with PC/3270 or PC400 sessions, the client application should start DDE conversation with PC/3270 or PC400. To start DDE conversation, specify *IBM3270* or *IBM5250* as an application name and *SessionA*, *SessionB*, ..., *SessionZ* as topic names in the DDE function (Initiate) in the client application.

Start Hot Link with Session Window

After starting DDE conversation, the client application can start the Advise function. Specify the following item names in the DDE function (Advise) in the client application to start Hot Link, which enables the automatic data update:

Item	Explanation	DDE function
CLOSE	Starts to intercept Window Close request	Start Close Intercept
KEYS	Starts to intercept keystrokes	Start Keystroke Intercept
PS *1 OIA TRIMRECT *1	Start to retrieve data of PS, OIA, or trim rectangle.	Start Session Advise

***1:** Parameter should be added

Stopping Hot Link with Session Window

To terminate the Advise function, the client application needs to stop the Advise function using the DDE function. Specify the following item names in the DDE function (Unadvise) in the client application to stop Hot Link, which enables the automatic data update:

Item	Explanation	DDE function
CLOSE	Stops to intercept Close request	Stop Close Intercept
KEYS	Stops to intercept keystrokes	Stop Keystroke Intercept
PS *1 OIA TRIMRECT *1	Stops to Advise function for the session	Stop Session Advise

***1:** Use the same parameter that are used when Start Session Advise was called

Terminating Session DDE Conversation

The client application should terminate the DDE conversation with PC/3270 or PC400 when completing the task. To terminate the conversation, use the DDE function (Terminate) in the client application.

PC/3270 and PC400 DDE Interface

This section describes the DDE functions that can be used from the other applications such as Microsoft Excel, Microsoft Word, and Microsoft Visual Basic.

DDE Functions for System Conversation

Function	Page
Initiate System Conversation	287
Get System Configuration	287
Get System Formats	288
Get System Status	288
Get System SysItems	288
Get System Topics	289
Terminate System Conversation	289

DDE Functions for Session Conversation

Function	Page
Initiate Session Conversation *1	287
Find Field	290
Get Operator Information Area	292
Get Partial Presentation Space	293
Get Presentation Space	293
Get Session Status	294
Get Trim Rectangle	294
Put Data to Presentation Space	295
Search for String	296
Session Execute Macro	296
Set Cursor Position	297
Terminate Session Conversation *2	302

DDE Function for Session Conversation (Hot Link)

Function	Page
Initiate Session Conversation (same as *1)	298
Start Close Intercept	298
Start Keystroke Intercept	299
Start Session Advise	299
Stop Close Intercept	301
Stop Keystroke Intercept	301
Stop Session Advise	301
Terminate Session Conversation (same as *2)	302

DDE Functions for System Conversation

The following DDE functions are provided for PC/3270 or PC400 system conversation.

Initiate System Conversation

The Initiate System Conversation function starts the system conversation. Only one client application can be connected to one system.

DDE parameter	*Value*
Topic	System

The client application should start DDE conversation using the DDE function (Initiate) with PC/3270 or PC400 application name (*IBM3270* or *IBM5250*) and the topic name (System).

Get System Configuration

The Get System Configuration function returns PC/3270 or PC400 support level and other system related values.

DDE parameter	*Value*
Item	SysCon

The client application can use the PC/3270 or PC400 DDE function by specifying the DDE function (Request) of the client application with the topic name (System).

PC/3270 or PC400 Response

PC/3270 or PC400 system returns PC/3270 or PC400 system configuration data item.

Returned Information: Refer to “Get System Configuration” on page 210 for details.

The possible reasons for PC/3270 or PC400 to not return the system configuration data item are:

The specified item name is not valid
A system error occurred

Get System Formats

The Get System Formats function returns a list of Windows Clipboard formats supported by PC/3270 or PC400.

DDE parameter	*Value*
Item	Formats

The client application can use the PC/3270 or PC400 DDE function by specifying the DDE function (Request) of the client application with the item name (Formats).

PC/3270 or PC400 Response

PC/3270 or PC400 returns a list of supported Windows Clipboard formats.

The possible reasons that cause PC/3270 or PC400 to not return the format data item are as follows:

The specified item name is not valid
A system error occurred

Get System Status

The Get System Status function returns the status of each configured PC/3270 or PC400 session.

DDE parameter	*Value*
Item	SysCon

The client application can use the PC/3270 or PC400 DDE function by specifying the DDE function (Request) of the client application with the item name (SysCon).

PC/3270 or PC400 Response

PC/3270 or PC400 returns a series of status information to each open session.

Returned Information: Refer to “Get System Status” on page 213 for details.

The possible reasons that cause PC/3270 or PC400 to not return the status data item are as follows:

The specified item name is not valid
A system error occurred

Get System SysItems

The Get System SysItems function returns a list of data items that can be used with the PC/3270 or PC400 system topic.

DDE parameter	*Value*
Item	SysItems

The client application can use the PC/3270 or PC400 DDE function by specifying the DDE function (Request) of the client application with the item name (SysItems).

PC/3270 or PC400 Response

PC/3270 or PC400 returns a list of PC/3270 or PC400 system topic data items. The following data items are supported by PC/3270 or PC400:

SysItems
Topics
Status
Formats
SysCon

The possible reasons that cause PC/3270 or PC400 to not return the system data item are as follows:

The specified item name is not valid
A system error occurred

Get System Topics

The Get System Topics function returns a list of active DDE topics that are supported by PC/3270 or PC400.

DDE parameter	*Value*
Item	Topics

The client application can use the PC/3270 or PC400 DDE function by specifying the DDE function (Request) of the client application with the item name (Topics).

PC/3270 or PC400 Response

The following topics are supported by PC/3270 or PC400:

System
SessionA, SessionB, ..., SessionZ

The possible reasons that cause PC/3270 or PC400 to not return the system data item are as follows:

The specified item name is not valid
A system error occurred

Terminate System Conversation

The Terminate System Conversation function terminates the system conversation. Use the DDE function (Terminate) to terminate the DDE conversation from the client application.

DDE Functions for Session Conversation

The following DDE functions are provided for PC/3270 or PC400 session conversation.

Initiate Session Conversation

The Initiate Session Conversation function starts a DDE conversation in the available session window. Only one client application can be connected to one session conversation.

DDE parameter	*Value*
Topic	SessionA, SessionB, ..., SessionZ

Parameter Value	Explanation
SessionA, SessionB, ..., SessionZ	"SessionA" implies a string combined "Session" and a session ID “A”, “B”, ..., “Z”.

The client application should start the DDE conversation by specifying the DDE function (Initiate) of the client application with the topic name (SessionA, SessionB, ..., SessionZ).

PC/3270 or PC400 Response

If a topic is not specified, PC/3270 or PC400 responds after confirming the following available topics:

System
SessionA, SessionB, ..., SessionZ

Find Field

The Find Field function passes the field information to the client application.

DDE parameter	*Value*
Item	FIELD (pos, "type")

Parameter	Value	Explanation
pos	NNNN	PS position
"type"	" " or "T "	This field.
	"P "	The previous field, either protected or unprotected.
	"N "	The next field, either protected or unprotected. The next protected field.
	"NP"	The next unprotected field.
	"NU"	The previous protected field.
	"PP"	The previous unprotected field.
	"PU"	

Note: The symbol represents a required blank.

The IBM Personal Communications AS/400 and 3270 Version 3.1 format is also supported.

The client application can use the PC/3270 or PC400 DDE function by specifying the DDE function (Request) of the client application with the above item name.

PC/3270 or PC400 Response

The following table shows the field information that PC/3270 returns:

Field	Returned Information	Explanation
Formatted/Unformatted	Formatted, Unformatted	PS is formatted or unformatted. If Unformatted is specified, no other field information will be returned.
Unprotected/Protected	N	0=Unprotected data field 1=Protected data field
A/N	N	0=Alphanumeric 1=Numeric
I/SPD	N	0=Normal intensity, undetectable 1=Normal intensity, detectable 2=High intensity, detectable 3=Nondisplay, undetectable
MDT	N	0=Field is not changed 1=Field is changed
Field start offset	NNNN	Field starts this field position
Field length	NNNN	Field length

The following table shows the field information that PC400 returns:

Field	Returned Information	Explanation
Formatted/Unformatted	Formatted, Unformatted	PS is formatted or unformatted. If Unformatted is specified, no other field information will be returned.
Field attribute	N	0 = Non–field attribute byte 1 = Field attribute byte
Visibility	N	0 = Nondisplay 1 = Display
Unprotected/Protected	N	0 = Unprotected data field 1 = Protected data field
Intensity	N	0 = Normal 1 = High
Field Type	N	0 = Alphanumeric: all characters allowed. 1 = Alphabet only: uppercase and lowercase letters, comma, period, hyphen, blank, and Dup key allowed. 2 = Numeric shift: automatic shift for numerics. 3 = Numeric only: numbers 0–9, comma, period, plus, minus, blank, and Dup key allowed. 5 = Digits only: numbers 0–9 and Dup key allowed. 6 = Magnetic stripe reader data only 7 = Signed numeric: numbers 0–9, plus, minus, and Dup key allowed.
MDT	N	0=Field is not changed 1=Field is changed
Field start offset	NNNN	Field starts this field position
Field length	NNNN	Field length

The possible reasons that cause PC/3270 or PC400 to not return the field information are as follows:

The specified item name is not valid

A system error occurred

Get Operator Information Area

The Get Operator Information Area function returns the OIA data information to the client application.

DDE parameter	*Value*
Item	OIA

The client application can use the PC/3270 or PC400 DDE function by specifying the DDE function (Request) of the client application with the item name (OIA).

PC/3270 or PC400 Response

The following chart shows the OIA information that PC/3270 or PC400 returns:

Offset	Returned Information	Meaning
0	ONLINE	Online, the screen is unowned
	LU-LU	LU-LU session owns the screen
	SSCP-LU	SSCP-LU session owns the screen
9	X	Input inhibit
	X MCHK	Machine check
	X CCHK	Communication check
	X PCHK	Program check
	X DNW	Device not working
	X BUSY	Printing
	X TWAIT	Terminal waiting
	X -S	Minus symbol
	X -f	Minus function
	X MUCH	Input too much
	X UA	Unauthorized operator
	X -fUA	Unauthorized operator Minus function
	X DEAD	The dead key combination is not valid
	X WRONG	Wrong position
	X SYSTEM	System waiting
	X II	Operator input error (PC400)
19	COMM	Communication error
25	MW	Message waiting (PC400)
36	APL	APL (PC/3270)
42	U	Uppercase
	NUM	Numeric
43	A	Caps lock
47	S	High-intensity, operator selectable
	I	High-intensity, field inherit
49	CS	Color, operator selectable
	CI	Color, field inherit
52	^	Insert mode
61	P-MAL	Printer malfunction
	P-PRN	Printer printing
	P-ASS	Printer assignment

The possible reasons that cause PC/3270 or PC400 to not return the OIA information are as follows:

The specified item name is not valid
A system error occurred

Get Partial Presentation Space

The Get Partial Presentation Space function returns whole or partial presentation space data to the client application.

DDE parameter *Value*
Item EPS (pos, len, bEOF)

Parameter	Value	Explanation
pos	NNNN	PS position
len	NNNN	PS length
bEOF	1 or 0	EOF switch **1** Yes **0** No

Note: Item of the IBM Personal Communications AS/400 and 3270 Version 3.1 format is also supported.

The client application can use the PC/3270 or PC400 DDE function by specifying the DDE function (Request) of the client application with the above item names.

PC/3270 or PC400 Response

The following chart shows the information PC/3270 or PC400 returns:

Field	Returned Information	Explanation
PS start position	NNNN	Specified by pos parameter
PS length	NNNN	Specified by len parameter
PS rows	NNNN	
PS columns	NNNN	
PS		PS data starts from this position

The possible reasons that cause PC/3270 or PC400 to not return the format data items are as follows:

The specified item name is not valid
A system error occurred

Get Presentation Space

The Get Presentation Space function returns presentation space data to the client application.

DDE parameter *Value*
Item PS

The client application can use the PC/3270 or PC400 DDE function by specifying the DDE function (Request) of the client application with the item name (PS).

PC/3270 or PC400 Response

The following chart shows the information PC/3270 or PC400 returns:

Field	Returned Information	Explanation
PS size	NNNN	
PS rows	NNNN	
PS columns	NNNN	
PS		PS data starts from this position

The possible reasons that cause PC/3270 or PC400 to not return the format data items are as follows:

The specified item name is not valid
A system error occurred

Get Session Status

The Get Session Status function returns the connected session status to the client application.

DDE parameter	*Value*
Item	SSTAT

The client application can use the PC/3270 or PC400 DDE function by specifying the DDE function (Request) of the client application with the item name (SSTAT).

PC/3270 or PC400 Response

Refer to “Get Session Status” on page 208 for the returned information.

The possible reasons that cause PC/3270 or PC400 to not return the format data items are as follows:

The specified item name is not valid
A system error occurred

Get Trim Rectangle

The Get Trim Rectangle function returns the presentation space area of the current (or specified) trim rectangle to the client application.

DDE parameter	*Value*
Item	TRIMRECT (row1, col1, row2, col2) TRIMRECT (pos1, pos2) TRIMRECT

Parameter	Value	Explanation
row1	NN	Top-left corner row of the trim rectangle
col1	NN	Top-left corner column of the trim rectangle
row2	NN	Bottom-right corner row of the trim rectangle
col2	NN	Bottom-right corner column of the trim rectangle
pos1	NNNN	PS position of the top-left corner of the trim rectangle
pos2	NNNN	PS position of the bottom-right corner of the trim rectangle

Note: Item of the IBM Personal Communications AS/400 and 3270 Version 3.1 format is also supported.

The current specified PS trim rectangle is used unless the client application specifies the PS trim rectangle in the parameter.

The client application can use the PC/3270 or PC400 DDE function by specifying the DDE function (Request) of the client application with the item name (TRIMRECT).

PC/3270 or PC400 Response

The information returned from PC/3270 or PC400 is as follows:

Field	Returned Information	Explanation
PS		PS data starts from this position.

The possible reasons that cause PC/3270 or PC400 to not return the trim rectangle items are as follows:

The specified item name is not valid
A system error occurred

Put Data to Presentation Space

The Put Data to Presentation Space function sends ASCII data string to write on the specified host presentation space.

DDE parameter *Value*
Item EPS (pos, bEOF)

Parameter	Value	Explanation
pos	NNNN	PS position to start writing the data
bEOF	1 or 0	EOF switch **1** Yes **0** No

Note: Item of the IBM Personal Communications AS/400 and 3270 Version 3.1 format is also supported.

The client application can use the PC/3270 or PC400 DDE function by specifying the DDE function (Poke) of the client application with the above item name.

PC/3270 or PC400 Response

The possible reasons that cause PC/3270 or PC400 to not accept the string data are as follows:

The specified item name is not valid
The PS position is not valid
The length is not valid
PS input inhibited
A system error occurred

Search for String

Using the Search for String function, the client application can check whether the specified strings exist within the specified presentation space area.

DDE parameter *Value*
Item STRING (pos, bDir, "string")

Parameter	Value	Explanation
pos	NNNN	PS start position of the string search
bDir	1 or 0	Search Direction **1** Forward **0** Backward
"string"		Search string Enclose the string including blanks with double quotation marks. To specify a double quotation mark within the string, enclose the double quotation mark with another set of double quotation marks. Example: *This is a double quotation" mark.* is specified as *"This is a double quotation""" mark."*.

Note: Item of the IBM Personal Communications AS/400 and 3270 Version 3.1 format is also supported.

The maximum length of the search string is 255.

The client application can use the PC/3270 or PC400 DDE function by specifying the DDE function (Request) of the client application with the above item names.

PC/3270 or PC400 Response

PC/3270 or PC400 returns the following information.

The information returned from PC/3270 or PC400 is as follows:

Field	Returned Information	Explanation
String start offset	NNNN, None	"None" is returned if the string is not found

The possible reasons that cause PC/3270 or PC400 to not return the string start position are as follows:

The specified item name is not valid
PS position is not valid or the string is too long
A system error occurred

Session Execute Macro

The Session Execute Macro function enables users to send commands and macro strings to PC/3270 or PC400.

Refer to “Session Execute Macro” on page 223 for details of commands and macro strings.

The client application can use the PC/3270 or PC400 DDE function by specifying the DDE function (Execute) of the client application.

PC/3270 or PC400 Response

A system error occurrence is a possible reason that PC/3270 or PC400 do not return the string start position.

Set Cursor Position

Using the Set Cursor Position function, the client application can set the cursor position in the session window.

DDE parameter	*Value*
Item	SETCURSOR
Data (Cursor position)	NNNN or Rn1Cn2

Parameter Value	Explanation
NNNN	PS offset
Rn1Cn2	Row/column **n1** PS position row **n2** PS position column

The client application can use the PC/3270 or PC400 DDE function by specifying the DDE function (Poke) of the client application with the above item names.

PC/3270 or PC400 Response

The possible reasons that cause PC/3270 or PC400 to not move the cursor to the specified PS position are as follows:

The specified item name is not valid
Cursor PS offset is not valid. (Have to be from 0 to PS size-1)
Cursor row value is not valid. (Have to be from 0 to PS row-1)
Cursor column value is not valid. (Have to be from 0 to PS column-1)
A system error occurred

Terminate Session Conversation

The Terminate Session Conversation function terminates the DDE conversation between the client application and PC/3270 or PC400.

Use the DDE function (Terminate) of the client application to terminate the DDE conversation.

DDE Function for Session Conversation (Hot Link)

The following DDE functions are provided for PC/3270 or PC400 session conversation with Hot Link connection.

Initiate Session Conversation

The Initiate Session Conversation function starts a DDE conversation with the available session window.

DDE parameter	*Value*
Topic	SessionA, SessionB, ..., SessionZ or SessA_xxxx, SessB_xxxx, ..., SessZ_xxxx

Note: If SessA_xxxx, SessB_xxxx, ..., SessZ_xxxx is used, the client application allows only Hot Link session conversation.

Parameter Value	Explanation
SessA_xxxx, SessB_xxxx, ..., SessZ_xxxx	String SessA_xxxx indicates session A (SessA_) with any user defined strings (xxxx). The length of the user-defined strings is not limited.

Specify the PC/3270 or PC400 application name and the above topic name in the DDE function (Initiate) of the client application to start a DDE conversation.

Start Close Intercept

Using the Start Close Intercept function, the client application can intercept the Close request generated by selecting the Close option from the Emulator session window. When this service is started, the client application receives the Close request event data.

DDE parameter	*Value*
Item	CLOSE

The client application can use the PC/3270 or PC400 DDE function by specifying the DDE function (Advise) of the client application with the above item name.

PC/3270 or PC400 Response

PC/3270 or PC400 returns the following information.

Returned Information: The information returned from PC/3270 or PC400 is as follows:

Field	Returned Information	Explanation
Number of PS close request	NNNN	When Close request is generated, the client application receives 0001.

The possible reasons that cause PC/3270 or PC400 to not start to Close intercept are as follows:

The specified item name is not valid
The Close intercept against the session has already started with the same topic name
A system error occurred

Start Keystroke Intercept

Using the Start Keystroke Intercept function, the client application can filter keystrokes that are entered by the terminal operator. When started, the keystrokes are intercepted and received by the client application.

DDE parameter	*Value*
Item	KEYS

The client application can use the PC/3270 or PC400 DDE function by specifying the DDE function (Advise) of the client application with the above item name.

PC/3270 or PC400 Response

Returned Information: The information returned from PC/3270 or PC400 is as follows:

Field	Returned Information	Explanation
Keys		Refer to Table 17 “SENDKEY Command List” of Chapter 8, “DDE Functions” “Session Execute Macro” on page 226.

The possible reasons that cause PC/3270 or PC400 to not start Keystroke Intercept are as follows:

The specified item name is not valid
The Keystroke Intercept against the session has already started with the same topic name
A system error occurred

Start Session Advise

The Start Session Advise function establishes the link between the client application and PC/3270 or PC400. As the data item is changed, the client application receives the changed data of the presentation space (PS), Operator Information Area (OIA), or trim rectangle (TRIMRECT).

DDE parameter	*Value*
Item	PS (pos, len, bCaseSen, "string") PS TRIMRECT (row1, col1, row2, col2) TRIMRECT (pos1, pos2) TRIMRECT OIA

The maximum length of the search string is 255.

Parameter	Value	Explanation
pos	NNNN	PS start position of the string search (PS offset)
len	NNNN	Length of the search string

Parameter	Value	Explanation
bCaseSen	1 or 0	Case sensitivity **1** Yes **0** No
"string"		Search string Enclose the string including blanks with double quotation marks. To specify a double quotation mark within the string, enclose the double quotation mark with another set of double quotation marks. Example: *This is a double quotation" mark.* is specified as *"This is a double quotation""" mark."*.

Note: Item of the IBM Personal Communications AS/400 and 3270 Version 3.1 format is also supported.

To receive a conditional advice when the presentation space is updated, the client application needs to set the advise conditions as well as the above parameter values. The above parameter values can be used when the PS is specified as the item name.

Parameter	Value	Explanation
row1	NN	Top-left corner row of the trim rectangle
col1	NN	Top-left corner column of the trim rectangle
row2	NN	Bottom-right corner row of the trim rectangle
col2	NN	Bottom-right corner column of the trim rectangle
pos1	NNNN	PS position of the top-left corner of the trim rectangle
pos2	NNNN	PS position of the bottom-right corner of the trim rectangle

Note: Item of the IBM Personal Communications AS/400 and 3270 Version 3.1 format is also supported.

The current specified PS trim rectangle is used unless the client application specifies the PS trim rectangle in the item name parameter. This parameter value can be used when TRIMRECT is specified as the item name.

The client application can use the PC/3270 or PC400 DDE function by specifying the DDE function (Advise) of the client application with the above item name.

PC/3270 or PC400 Response

PC/3270 or PC400 returns the following information.

Returned Information: Refer to “Get Partial Presentation Space” on page 293, “Get Operator Information Area” on page 292, and “Get Trim Rectangle” on page 294.

The possible reasons that cause PC/3270 or PC400 to not start Advise are as follows:

The specified item name is not valid
The Advise against the session has already started with the same topic name
A system error occurred

Stop Close Intercept

Using the Stop Close Intercept function, the client application stops intercepting the close requests.

DDE parameter	*Value*
Item	CLOSE

The client application can use the PC/3270 or PC400 DDE function by specifying the DDE function (Unadvise) of the client application with the above item name.

PC/3270 or PC400 Response

The possible reasons that cause PC/3270 or PC400 to not stop close intercept are as follows:

Advise has not been started
A system error occurred

Stop Keystroke Intercept

Using the Stop Close Intercept function, the client application stops intercepting the keystrokes.

DDE parameter	*Value*
Item	KEYS

The client application can use the PC/3270 or PC400 DDE function by specifying the DDE function (Unadvise) of the client application with the above item name.

PC/3270 or PC400 Response

The possible reasons that cause PC/3270 or PC400 to not stop keystroke intercept are as follows:

The specified item name is not valid
Advise has not been started
A system error occurred

Stop Session Advise

The Stop Session Advise function closes the link between the client application and PC/3270 or PC400.

DDE parameter	*Value*
Item	PS (pos, len, bCaseSen, "string") PS TRIMRECT (row1, col1, row2, col2) TRIMRECT (pos1, pos2) TRIMRECT OIA

The maximum length of the search string is 255.

The item name must be the same item name that was used when Start Session Advise was called.

The client application can use the PC/3270 or PC400 DDE function by specifying the DDE function (Unadvise) of the client application with the above item name.

PC/3270 or PC400 Response

The possible reasons that cause PC/3270 or PC400 to not stop Advise are as follows:

The specified item name is not valid
Advise has not been started
A system error occurred

Terminate Session Conversation

The Terminate Session Conversation function terminates the DDE conversation between the client application and the PC/3270 or PC400 session.

Use the DDE function (Terminate) of the client application to terminate the DDE conversation.

Visual Basic Sample Program

Following is a sample program with Visual Basic:

Note: The sample program below is simplified and differs from the actual sample file provided.

```
'/                                    /
'/                                                    /
'/        System  conversation                    /
'/                                                    /
'/                                    /

'
'
'         Initiate System Conversation
'
'
'
'   Start DDE Conversation with system
'
Sub Command1_Click ()
On Error GoTo ErrHandlerInit
     Dim COLD As Integer
     COLD = 2
     FunctionComp& = True

    DoEvents
     Text1.LinkTopic = "IBM525 |System"
    Text1.LinkMode    =  COLD

     If FunctionComp&= False Then
          MsgBox "Error was occurred", 48, "DDE sample"
    End  If
    Exit  Sub

ErrHandlerInit:
     FunctionComp& = False
    Resume  Next
End Sub

'-- note ------------------------------------------
'
' If you use VisualBasic Version2. , use "DoEvents"
' function before starting DDE conversation by
' calling "LinkMode" function.
'
'-------------------------------------------------
'
'
'         Get System Format
'
'
'
'     Request a list of PC/327  or PC4
'    Clipboard  Format
'
Sub Command2_Click ()
On Error GoTo ErrHandler
     FunctionComp& = True

     Text1.LinkItem = "Formats"
    Text1.LinkRequest

     If FunctionComp&= False Then
          MsgBox "Error was occurred", 48, "DDE sample"
    End  If
    Exit  Sub

ErrHandler:
     FunctionComp& = False
```

```
    Resume  Next
End Sub

'
'
'          Get System Status
'
'
'
'     Requests each PC/327  or PC4
'     Session  Status
'
Sub Command2_Click ()
On Error GoTo ErrHandler
     FunctionComp& = True

     Text1.LinkItem = "Status"
    Text1.LinkRequest

     If FunctionComp&= False Then
          MsgBox "Error was occurred", 48, "DDE sample"
    End  If
    Exit  Sub

ErrHandler:
     FunctionComp& = False
    Resume  Next
End Sub

'
'
'          Get System Configuration
'
'
'
'     Requests PC/327  or PC4
'     System Configuration Values
'
Sub Command2_Click ()
On Error GoTo ErrHandler
     FunctionComp& = True

     Text1.LinkItem = "SysCon"
    Text1.LinkRequest

     If FunctionComp&= False Then
          MsgBox "Error was occurred", 48, "DDE sample"
    End  If
    Exit  Sub

ErrHandler:
     FunctionComp& = False
    Resume  Next
End Sub

'
'
'          Get System SysItems
'
'
'
'     Requests a list of Data Items for
'     PC/327  or PC4   System Conversation
'
Sub Command2_Click ()
On Error GoTo ErrHandler
     FunctionComp& = True

     Text1.LinkItem = "SysItems"
    Text1.LinkRequest

     If FunctionComp&= False Then
```

```
        MsgBox "Error was occurred", 48, "DDE sample"
    End  If
    Exit  Sub

ErrHandler:
     FunctionComp& = False
    Resume  Next
End Sub

'
'
'        Get System Topics
'
'
'
'     Requests a list of PC/327  or PC4
'    Topics
'
Sub Command2_Click ()
On Error GoTo ErrHandler
     FunctionComp& = True

     Text1.LinkItem = "Topics"
    Text1.LinkRequest

     If FunctionComp&= False Then
        MsgBox "Error was occurred", 48, "DDE sample"
    End  If
    Exit  Sub

ErrHandler:
     FunctionComp& = False
    Resume  Next
End Sub

'
'
'        Terminate System Conversation
'
'
'
'   Terminates DDE Conversation with system
'
Sub Command3_Click ()
On Error GoTo ErrHandler
     Dim NONE As IntegerTerm
     NONE =
     FunctionComp& = True

     Text1.LinkMode = NONE

     If FunctionComp&= False Then
        MsgBox "Error was occurred", 48, "DDE sample"
    End  If
    Exit  Sub

ErrHandlerTerm:
     FunctionComp& = False
    Resume  Next
End Sub

'/                                  /
'/                                                  /
'/        Session  conversation                  /
'/                                                  /
'/                                  /

'
'
'        Initiate Session Conversation
'
'
```

```
'
'    Initiate DDE Conversation with system
'
Sub Command1_Click ()
On Error GoTo ErrHandlerInit
     Dim COLD As Integer
     COLD = 2
     FunctionComp& = True

    DoEvents
     Text1.LinkTopic = "IBM525 |SessionA"
     Text1.LinkMode = COLD

     If FunctionComp&= False Then
          MsgBox "Error was occurred", 48, "DDE sample"
    End  If
    Exit  Sub

ErrHandlerInit:
     FunctionComp& = False
    Resume  Next
End Sub

'-- note ---------------------------------------------
'
' If you use VisualBasic Version2. , use "DoEvents"
' function before starting DDE conversation by
' calling "LinkMode" function.
'
'-----------------------------------------------------
'
'
'         Find  Field
'
'
'
'     Requests 1   Field Information of PS Position
'
Sub Command2_Click ()
On Error GoTo ErrHandler
     FunctionComp& = True

     Text1.LinkItem = "FILED(1  ,""     "")"
    Text1.LinkRequest

     If FunctionComp&= False Then
          MsgBox "Error was occurred", 48, "DDE sample"
    End  If
    Exit  Sub

ErrHandler:
     FunctionComp& = False
    Resume  Next
End Sub

'
'
'          Get Operator Information Area
'
'
'
'     Requests OIA Data
'
Sub Command2_Click ()
On Error GoTo ErrHandler
     FunctionComp& = True

     Text1.LinkItem = "OIA"
    Text1.LinkRequest

     If FunctionComp&= False Then
          MsgBox "Error was occurred", 48, "DDE sample"
```

```
    End  If
    Exit  Sub

ErrHandler:
     FunctionComp& = False
    Resume  Next
End Sub

'
'
'         Get Partial Presentation Space
'
'
'
'     Requests PS Data Bytes from PS Position from 1   to 1
'
Sub Command2_Click ()
On Error GoTo ErrHandler
     FunctionComp& = True

     Text1.LinkItem = "EPS(1  ,1   ,1)"
    Text1.LinkRequest

     If FunctionComp&= False Then
          MsgBox "Error was occurred", 48, "DDE sample"
    End  If
    Exit  Sub

ErrHandler:
     FunctionComp& = False
    Resume  Next
End Sub

'
'
'         Get Presentation Space
'
'
'
'     Requests PS Data
'
Sub Command2_Click ()
On Error GoTo ErrHandler
     FunctionComp& = True

     Text1.LinkItem = "PS"
    Text1.LinkRequest

     If FunctionComp&= False Then
          MsgBox "Error was occurred", 48, "DDE sample"
    End  If
    Exit  Sub

ErrHandler:
     FunctionComp& = False
    Resume  Next
End Sub

'
'
'         Get Session Status
'
'
'
'     Requests Session Connection Status
'
Sub Command2_Click ()
On Error GoTo ErrHandler
     FunctionComp& = True

     Text1.LinkItem = "SSTAT"
    Text1.LinkRequest
```

```
    If FunctionComp&= False Then
        MsgBox "Error was occurred", 48, "DDE sample"
    End  If
    Exit  Sub

ErrHandler:
    FunctionComp& = False
    Resume  Next
End Sub

'
'
'         Get Trim Rectangle
'
'
'
'     Requests PS Data in Current Specified Trim Rectangle
'
Sub Command2_Click ()
On Error GoTo ErrHandler
    FunctionComp& = True

    Text1.LinkItem = "TRIMRECT"
    Text1.LinkRequest

    If FunctionComp&= False Then
        MsgBox "Error was occurred", 48, "DDE sample"
    End  If
    Exit  Sub

ErrHandler:
    FunctionComp& = False
    Resume  Next
End Sub

'
'
'         Put Data to Presentation Space
'
'
'
'     Writes string "Hello, World!" from PS Position 2
'
Sub Command2_Click ()
On Error GoTo ErrHandler
    FunctionComp& = True

    Text1.Text         = "Hello, World!"
    Text1.LinkItem = "EPS(2  ,1)"
    Text1.LinkPoke

    If FunctionComp&= False Then
        MsgBox "Error was occurred", 48, "DDE sample"
    End  If
    Exit  Sub

ErrHandler:
    FunctionComp& = False
    Resume  Next
End Sub

'
'
'         Search for String
'
'
'
'     Search forward for string "Hello!" from PS Position 1
'
Sub Command2_Click ()
On Error GoTo ErrHandler
```

```
    FunctionComp& = True

   Text1.LinkItem    =  "STRING(1,1,""Hello!"")"
   Text1.LinkRequest

    If FunctionComp&= False Then
        MsgBox "Error was occurred", 48, "DDE sample"
   End  If
   Exit  Sub

ErrHandler:
    FunctionComp& = False
   Resume  Next
End Sub

'
'
'         Session Execute Macro
'
'
'
'     Maximize the Session
'
Sub Command2_Click ()
On Error GoTo ErrHandler
    FunctionComp& = True

   Text1.LinkExecute  "[WINDOW(MAXIMIZE)]"

    If FunctionComp&= False Then
        MsgBox "Error was occurred", 48, "DDE sample"
   End  If
   Exit  Sub

ErrHandler:
    FunctionComp& = False
   Resume  Next
End Sub

'
'
'         Set Cursor Position
'
'
'
'     Set Cursor Position (Row,Column) = (1,1)
'
Sub Command2_Click ()
On Error GoTo ErrHandler
    FunctionComp& = True

   Text1.Text         =  "R1C1"
    Text1.LinkItem = "SETCURSOR"
   Text1.LinkPoke

    If FunctionComp&= False Then
        MsgBox "Error was occurred", 48, "DDE sample"
   End  If
   Exit  Sub

ErrHandler:
    FunctionComp& = False
   Resume  Next
End Sub

'
'
'         Terminate Session Conversation
'
'
'
'   Terminate DDE Conversation with session
```

```
'
Sub Command3_Click ()
On Error GoTo ErrHandlerTerm
     Dim NONE As Integer
     NONE =
     FunctionComp& = True

     Text1.LinkMode = NONE

     If FunctionComp&= False Then
          MsgBox "Error was occurred", 48, "DDE sample"
    End  If
    Exit  Sub

ErrHandlerTerm:
     FunctionComp& = False
    Resume  Next
End Sub

'/                                    /
'/                                                                  /
'/          Session conversation(Hot Link)                    /
'/                                                                  /
'/                                    /

'
'
'          Start Close Intercept
'
'
'
'    Start Intercepting Close request
'
Sub Command1_Click ()
On Error GoTo ErrHandlerInit
     Dim HOT As Integer
     HOT = 1
     FunctionComp& = True

    DoEvents
     Text1.LinkTopic = "IBM525 |SessionA"
    Text1.LinkItem    =  "CLOSE"
    Text1.LinkMode    =  HOT

     If FunctionComp&= False Then
          MsgBox "Error was occurred", 48, "DDE sample"
    End  If
    Exit  Sub

ErrHandlerInit:
     FunctionComp& = False
    Resume  Next
End Sub

'-- note ------------------------------------------
'
' If you use VisualBasic Version2. , use "DoEvents"
' function before starting DDE conversation by
' calling "LinkMode" function.
'
'-------------------------------------------------
'
'
'          Start Keystroke Intercept
'
'
'
'    Start Intercepting Keystrokes
'
Sub Command1_Click ()
On Error GoTo ErrHandlerInit
     Dim HOT As Integer
```

```
     HOT = 1
     FunctionComp& = True

    DoEvents
     Text1.LinkTopic = "IBM525 |SessionA"
    Text1.LinkItem    =  "KEYS"
    Text1.LinkMode    =  HOT

     If FunctionComp&= False Then
          MsgBox "Error was occurred", 48, "DDE sample"
    End  If
    Exit  Sub

ErrHandlerInit:
     FunctionComp& = False
    Resume  Next
End Sub

'-- note ---------------------------------------------
'
' If you use VisualBasic Version2. , use "DoEvents"
' function before starting DDE conversation by
' calling "LinkMode" function.
'
'----------------------------------------------------
'
'
'          Start Session Advise(PS)
'
'
'
'    Receives PS Data when updated
'    (only when "Hello!" is displayed from PS Position 1)
'
Sub Command1_Click ()
On Error GoTo ErrHandlerInit
     Dim HOT As Integer
     HOT = 1
     FunctionComp& = True

    DoEvents
     Text1.LinkTopic = "IBM525 |SessA_PS"
    Text1.LinkItem    =  "PS(1,6,1,""Hello!"")"
    Text1.LinkMode    =  HOT

     If FunctionComp&= False Then
          MsgBox "Error was occurred", 48, "DDE sample"
    End  If

    Exit  Sub

ErrHandlerInit:
     FunctionComp& = False
    Resume  Next
End Sub

'-- note ---------------------------------------------
'
' If you use VisualBasic Version2. , use "DoEvents"
' function before starting DDE conversation by
' calling "LinkMode" function.
'
'----------------------------------------------------
'
'
'          Start Session Advise(TRIMRECT)
'
'
'
'    Receives PS Data in Trim Rectangle when PS Data in Trim Rectangle
'    specified by R1C1:R2 C4  is changed
'
```

```
Sub Command1_Click ()
On Error GoTo ErrHandlerInit
     Dim HOT As Integer
     HOT = 1
     FunctionComp& = True

    DoEvents
     Text1.LinkTopic = "IBM525 |SessA_TRIMRECT"
    Text1.LinkItem    =  "TRIMRECT(1,1,2 ,4 )"
    Text1.LinkMode    =  HOT

     If FunctionComp&= False Then
          MsgBox "Error was occurred", 48, "DDE sample"
    End  If
    Exit  Sub

ErrHandlerInit:
     FunctionComp& = False
    Resume  Next
End Sub

'-- note --------------------------------------------
'
' If you use VisualBasic Version2. , use "DoEvents"
' function before starting DDE conversation by
' calling "LinkMode" function.
'
'---------------------------------------------------
'
'
'          Start Session Advise(OIA)
'
'
'
'    Receives OIA Data when changed
'
Sub Command1_Click ()
On Error GoTo ErrHandlerInit
     Dim HOT As Integer
     HOT = 1
     FunctionComp& = True

    DoEvents
     Text1.LinkTopic = "IBM525 |SessA_OIA"
    Text1.LinkItem    =  "OIA"
    Text1.LinkMode    =  HOT

     If FunctionComp&= False Then
          MsgBox "Error was occurred", 48, "DDE sample"
    End  If
    Exit  Sub

ErrHandlerInit:
     FunctionComp& = False
    Resume  Next
End Sub

'-- note --------------------------------------------
'
' If you use VisualBasic Version2. , use "DoEvents"
' function before starting DDE conversation by
' calling "LinkMode" function.
'
'---------------------------------------------------
'
'
'          Terminate Session Conversation(Hot Link)
'
'
'
'    Terminate DDE Conversation with session (Hot Link)
'
```

```
Sub Command3_Click ()
On Error GoTo ErrHandlerTerm
     Dim NONE As Integer
     NONE =
     FunctionComp& = True

     Text1.LinkMode = NONE

     If FunctionComp&= False Then
          MsgBox "Error was occurred", 48, "DDE sample"
    End  If
    Exit  Sub

ErrHandlerTerm:
     FunctionComp& = False
    Resume  Next
End Sub
```

DDE Functions with Visual Basic

Chapter 10. DOS Mode EHLLAPI Support in a Windows DOS Prompt for PC/3270 Version 1.01/2.0 for DOS

PC/3270 Version 4.1 for Windows supports DOS mode EHLLAPI for IBM Personal Communications/3270 Version 4.0 for DOS. The DOS application program interacting with 3270 session through DOS mode EHLLAPI can run in a Windows DOS prompt with PC/3270 Version 4.1 for Windows.

PC/3270 Windows mode Version 4.1 emulates the DOS mode EHLLAPI functions with these restrictions;

- Supports only Enhances Mode
- Trace facility is not supported

The following EHLLAPI functions are supported in a Windows DOS prompt.

- Connect Presentation Space (1)
- Disconnect Presentation Space (2)
- Send Key (3)
- Wait (4)
- Copy Presentation Space (5)
- Search Presentation Space (6)
- Query Cursor Location (7)
- Set Session Parameters (9)
- Query Sessions (10)
- Reserve (11)
- Release (12)
- Copy OIA (13)
- Query Field Attribute (14)
- Copy String to Presentation Space (15)
- Pause (18)
- Query System (20)
- Reset System (21)
- Query Session Status (22)
- Start Host Notification (23)
- Query Host Update (24)
- Stop Host Notification (25)
- Search Field (30)
- Find Field Position (31)
- Find Field Length (32)
- Copy String to Field (33)
- Copy Field to String (34)

© Copyright IBM Corp. 1996

Set Cursor (40)

Start Close Intercept (41)

Query Close Intercept (42)

Stop Close Intercept (43)

Start Keystroke Intercept (50)

Get Key (51)

Post Intercept Status (52)

Stop Keystroke Intercept (53)

Send File (90)

Receive File (91)

Convert Position or Convert RowCol (99)

Connect For Structured Fields (120)

Disconnect From Structured Fields (121)

Query Communication Buffer Size (122)

Allocate Communication Buffer (123)

Free Communication Buffer (124)

Get Request Completion (125)

Read Structured Field (126)

Write Structured Field (127)

A DOS application program can run in Windows DOS Prompt. To start Windows DOS Prompt, modify the PIF file (DOSPRMPT.PIF) with Windows PIF Editor. Open the PIF file and check the execution.

To use Structured Field functions:

1. Connect For Structured Field
2. Disconnect From Structured Fields(121),
3. Query Communication Buffer
4. Allocate Communication Buffer(123),
5. Free Communication Buffer
6. Get Request Completion(125)
7. Read Structured Field(126), and
8. Write Structured Field(127)

use the PCSDHLL.PIF file in place of the default DOSPRMPT.PIF file.

The total communication buffer size (kilo bytes) should be specified to PCSDHLL.PIF. Open PCSDHLL.PIF with the Windows PIF Editor and set the size (1 - 35, Default is 35) into Optional Parameter section.

To start DOS application automatically, edit the PCSDHLL.BAT file, delete the COMMAND.COM /E:2048 line and write a DOS application program name with a full path.

PCSDHLL.PIF and PCSDHLL.BAT files are copied into the same directory where PC/3270 Version 4.1 for Windows is installed.

EHLLAPI Support in Windows DOS Prompt

Chapter 11. Server Requester Programming Interface Support

IBM Personal Communications supports the server requester programming interface (SRPI) function as follows:

1. The SRPI interface is the same function call as an API call and the parameters as that for OS/2 EE V1.3, except for multiple, concurrent SRPI requester applications to the same 3270 logical terminal session.
2. The return codes for errors or successful operations are the same as for OS/2 EE V1.3.
3. The SRPI interface is usable through a host connect of the emulator in all modes (except asynchronous and Control Unit Terminal connection). This connection is possible when physical connection to the host is over a token ring or a coaxial cable, or SNA or non-SNA protocols.
4. If a call is made to the SRPI interface and there is no response from the host due to a communications failure, then an associated error is returned to the caller.
5. SRPI and EHLLAPI are capable of concurrent operations.
6. SRPI is only supported for C requester.
7. Server Alias is not supported.
8. The 3270 screen update notify is not supported.
9. PC400 does not support this function.

SRPI API Interface

The SRPI API interface contains two function calls:

```
void pascal far init_send_req_parms(UERCPRB far  );
long int far send_request(UERCPRB far  );
```

© Copyright IBM Corp. 1996

EHLLAPI Support in Windows DOS Prompt

Appendix A. EHLLAPI Messages

COMHLL01 3270PCS is not installed

Explanation: The terminal emulator is not installed. **User Response**: Install the terminal emulator.

COMHLL02 PCSHLL is already installed

Explanation: The EHLLAPI program is installed. **User Response**: You do not need to install it again.

COMHLL03 Illegal parameter, PCSHLL is not installed

Explanation: EHLLAPI uses six parameters whose values are established by the customization program. One or more of the parameters in your customization batch file, where the parameters are specified, have been altered and cannot be used. **User Response**: Run the customization setup program again.

COMHLL04 PCSHLL cannot open link with 3270PCS

Explanation: For PCSHLL to be able to establish a link with 3270PCS, at least one host session logical terminal must be established. **User Response**: Establish a host connection with at least one logical terminal.

© Copyright IBM Corp. 1996

EHLLAPI Messages

Appendix B. EHLLAPI Customized Language Interface Modules

This chapter is for the DOS mode only (not applicable to Windows mode).
The PC/3270 emulator high-level language application program interface (EHLLAPI) is designed to allow you to develop support for languages and features that might be unique to your environment. By using a language interface module (LIM) as a bridge between the application program and the primary interface program, you can:

- Write a LIM for any language you want to support
- Use an existing LIM as a gateway to other external functions
- Change the format of the function calls or parameter lists to better suit your environment

The most basic step in getting ready to write your own LIM is to understand how the parts of EHLLAPI work together. Refer to Figure 3 on page 324 as you read this information.

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

1. Your EHLLAPI program calls the appropriate LIM.
2. The LIM takes your program's parameters and builds a parameter control block (PCB).
3. The LIM passes this control block and your EHLLAPI request to the resident interface module (PCSHLL.EXE) by means of a software interrupt hex 7F.
4. EHLLAPI interprets the information and passes the request onto the PC/3270 Emulation Program.
5. The PC/3270 Emulation Program performs (or attempts) the task.
6. After the task is completed (or attempted), EHLLAPI places a return code in the parameter control block and returns control to the calling LIM.
7. The LIM then returns control to your EHLLAPI application program.

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 language-unique data formats
- Construct a control block for the primary program
- Call (using an interrupt) PCSHLL.EXE
- Receive control back from PCSHLL.EXE
- Pass back necessary parameters and return control to the calling application

© Copyright IBM Corp. 1996

Customized LIMs

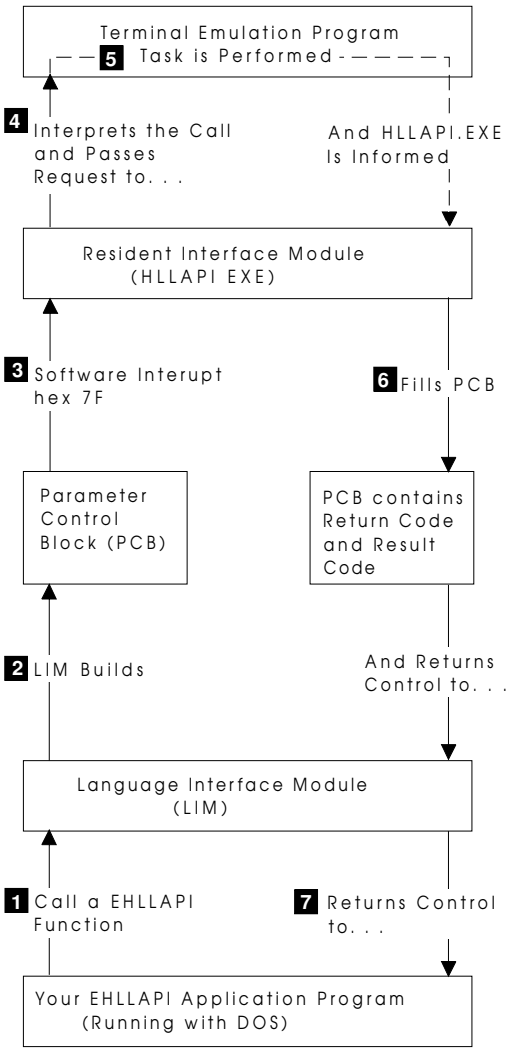


Figure 3. The Parts of PC/3270 EHLLAPI

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

Support for additional external functions

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 want to consolidate several of your subroutines into one function package, you can write an extended LIM for this purpose.

A simple technique is to reserve a range of function codes for each subroutine package. The LIM then examines the function code being invoked and calls the appropriate subroutine package. You might need to make adjustments in the number and type of parameters passed to EHLLAPI.

Revised Call Support for the Personal Computer

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

For example, some calls to EHLLAPI functions require a presentation space short name. As an ease-of-use feature, you might want to redesign the function calls to automatically insert the PSID.

Other revisions 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

Support for different data formats

Writing A 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, set register AX to hex 0104. Also, set register BX to 0.

Verify that the EHLLAPI module is loaded. Check to see if an interrupt vector has been established for the interface's PCSHLL.EXE module. The parameter control block has the following general format:

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

These rules apply in building the parameter control block:

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

The function code must be a 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 field length contains the actual length of the user's string, if the language you are using to write your EHLLAPI 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 used only 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, make the default a value of 25000.

There are special considerations for Storage Manager (17) function calls. The internal interface to the storage manager is different from the regular EHLLAPI 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:

Register	Explanation
On **Call**	
AX	0104H (mandatory)
BX	rrAAH where rr is the request code. Valid input is binary 01 for Get Storage subfunction 02 for Free Storage subfunction 03 for Query Free Storage subfunction 04 for Free All Storage subfunction
CX	Storage size for Get Storage subfunction, or Storage block ID for Free Storage subfunction
On **Return**	
AX	Storage segment address for Get Storage subfunction
BX	Storage offset address for Get Storage subfunction
CX	Storage Block ID for Get Storage subfunction
DX	Return Code value for all four subfunctions

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

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

Note that EHLLAPI passes the string data back directly, but the return code is passed back to the parameter control block. 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.

Appendix C. EHLLAPI Sample Programs

This appendix contains sample EHLLAPI programs for the DOS mode. The tables appearing before the sample programs list the functions used in the sample. The *Personal Communications/3270 for Windows* diskette 2 (diskette 3 for the 5.25-inch) contains a sample EHLLAPI program. All the files for the sample begin with HLL.

IBM BASIC Sample Program

Function Name	Function Number
Connect	1
Disconnect	2
Send Key	3
Wait	4
Search Presentation Space	6
Query Cursor Location	7
Set Session Parameters	9
Reserve	11
Release	12
Pause	18
Query System	20
Reset System	21
Start Host Notification	23
Query Host Update	24
Stop Host Notification	25
Find Field Position	31
Find Field Length	32
Copy String to Field	33
Set Cursor	40
Receive	91
Convert Position or RowCol	99

```
'
'
'                          SAMPLE EHLLAPI PROGRAM
'
'       Author:    IBM
'       Date Written:    1988
'       For use with PC/327
'
'       Certain VM/37  keywords used below may vary in
'            function depending on system configuration.
'            A host presentation space size of 192  is
'            assumed.
'
'
'                              MAIN PROGRAM
```

© Copyright IBM Corp. 1996

```
'
declare sub hllcbas(request%, dstring$, dlength%, retcode%)

'
'      Reset EHLLAPI.
'
request% = 21
expectedrc% =
gosub callhllapi

'
'      Establish connection to the host presentation space.
'
request% = 1
dstring$ = "A"
expectedrc% =
gosub callhllapi

'
'      Send the system and user ids to the host.
'
request% = 3
dstring$ = "systemid userid@E"
dlength% = 17
expectedrc% =
gosub callhllapi

'
'      Has CP asked for my password?    If not, wait for a host
'         update and check again.    Try this twice; if it still
'         hasn't asked, the system is either down or very slow.
'         In this case, just terminate.
'
request% = 6
dstring$ = "Enter your password"
dlength% = 19
expectedrc% = -1
gosub callhllapi

if (retcode% <>  ) then
      request% = 23
      dstring$ = "AP        "
      dlength% = 256
      expectedrc% =
     gosub  callhllapi

      request% = 9
      dstring$ = "IPAUSE"
      dlength% = 6
      expectedrc% =
     gosub  callhllapi

      searchrc% = 24
      i% = 1

      while ((i% <= 2) and (searchrc% <>  ))
           request% = 18
           dlength% = 1
```

```
            expectedrc% = -1
           gosub  callhllapi

            request% = 24
            dstring$ = "A"
            expectedrc% = -1
           gosub  callhllapi

            request% = 6
            dstring$ = "Enter your password"
            dlength% = 19
            expectedrc% = -1
           gosub  callhllapi
            searchrc% = retcode%

            i% = i% + 1
       wend

       if (searchrc% <>  ) then
            print "CP never asked for a password - sample terminating"
           system
           end
       endif
endif

'
'      Stop host notification and reset 'pause'.
'
request% = 25
dstring$ = "A"
expectedrc% =
gosub callhllapi

request% = 9
dstring$ = "FPAUSE"
dlength% = 6
expectedrc% =
gosub callhllapi

'
'      Send the password & enter.
'
request% = 3
dstring$ = "password@E"
dlength% = 1
expectedrc% =
gosub callhllapi

'
'      Keep clearing the host presentation space until the
'         ready prompt (R;) is found.
'
request% = 9
dstring$ = "SRCHFROM"
dlength% = 8
expectedrc% =
gosub callhllapi
```

```
searchrc% = 1
while (searchrc% <>  )
      request% = 6
      dstring$ = "MORE"
      dlength% = 4
      pspos% = 19
      retcode% = pspos%
      expectedrc% = -1
     gosub  callhllapi

      if (retcode% =  ) then
           request% = 3
           dstring$ = "@C"
           dlength% = 2
           expectedrc% =
          gosub  callhllapi
     endif

      request% = 6
      dstring$ = "HOLDING"
      dlength% = 7
      pspos% = 19
      retcode% = pspos%
      expectedrc% = -1
     gosub  callhllapi

      if (retcode% =  ) then
           request% = 3
           dstring$ = "@C"
           dlength% = 2
           expectedrc% =
          gosub  callhllapi
     endif

      request% = 6
      dstring$ = "R;"
      dlength% = 2
      pspos% = 1
      retcode% = pspos%
      expectedrc% = -1
     gosub  callhllapi

      searchrc% = retcode%
wend

'
'      Lock the host keyboard until processing has completed.
'
request% = 11
expectedrc% =
gosub callhllapi

'
'      Convert the beginning of the VM/CMS input area to a host
'          presentation space position.
'
request% = 99
dstring$ = "AR"
```

```
dlength% = 23
pspos% = 1
retcode% = pspos%
expectedrc% = -1
gosub callhllapi

clen% = retcode%

'
'       Where is the cursor now?
'
request% = 7
expectedrc% =
gosub callhllapi

'
'       Set the cursor to the beginning of the input area if
'          it's not already there.
'
if (dlength% <> clen%) then
       request% = 4
       pspos% = clen%
       retcode% = pspos%
       expectedrc% =
      gosub  callhllapi
endif

'
'       Execute filelist on the host.
'
request% = 3
dstring$ = "FILELIST@E"
dlength% = 1
expectedrc% =
gosub callhllapi

'
'       Wait for the command to complete...
'
request% = 4
expectedrc% =
gosub callhllapi

'
'       Is the file "profile exec" displayed in the list?
'
request% = 6
dstring$ = "PROFILE    EXEC"
dlength% = 13
pspos% = 161
retcode% = pspos%
expectedrc% =
gosub callhllapi

'
'       Find the beginning of the input area in filelist.
'
request% = 31
```

```
pspos% = 18
retcode% = pspos%
dstring$ = "T "
expectedrc% =
gosub callhllapi

fieldpsposition% = dlength%

'
'      Find the length of the input area.
'
request% = 32
dstring$ = "T "
pspos% = 18
retcode% = pspos%
expectedrc% =
exlen% = 139
gosub callhllapi

fieldlength% = dlength%

'
'      Block normal messages on VM.
'
request% = 33
'   set the data string to the size of the input area
dstring$ = space$(fieldlength%)
mid$(dstring$,1) = "SET MSG OFF"
dlength% = fieldlength%
retcode% = fieldpsposition%
expectedrc% =
gosub callhllapi

request% = 3
dstring$ = "@E"
dlength% = 2
expectedrc% =
gosub callhllapi

'
'      Suppress file transfer messages.
'
request% = 9
dstring$ = "QUIET"
dlength% = 5
expectedrc% =
gosub callhllapi

'
'      Prepare for file transfer by disconnecting from the
'         host presentation space.
'
request% = 2
expectedrc% =
gosub callhllapi
'
'      Download "profile exec".
'
```

```
request% = 91
dstring$ = "profile.exe a:profile exec a (ascii crlf"
dlength% = 4
'   drive number = 4 (D)
retcode% = 4
expectedrc% = 3
gosub callhllapi

'
'       Restore file transfer messages to the foreground session.
'
request% = 9
dstring$ = "NOQUIET"
dlength% = 7
expectedrc% =
gosub callhllapi

'
'       Reconnect to the host presentation space.
'
request% = 1
dstring$ = "A"
expectedrc% =
gosub callhllapi

'
'       Enable normal messages on VM.
'
request% = 33
dstring$ = space$(fieldlength%)
mid$(dstring$,1) = "SET MSG ON"
dlength% = fieldlength%
retcode% = fieldpsposition%
expectedrc% =
gosub callhllapi

request% = 3
dstring$ = "@E"
dlength% = 2
expectedrc% =
gosub callhllapi

'
'       Leave filelist.
'
request% = 3
dstring$ = "@3"
dlength% = 2
expectedrc% =
gosub callhllapi

'
'       Unlock the host keyboard.
'
request% = 12
expectedrc% =
gosub callhllapi
```

```
system
end


callhllapi:
'
'      Call the hllapi LIM and verify the return code.
'
call hllcbas(request%, dstring$, dlength%, retcode%)

'   system error
if ((retcode% = 9) and (request% <> 99)) then
       request% = 2
       dstring$ = SPACE$(35)
       expectedrc% =
       call hllcbas(request%, dstring$, dlength%, retcode%)
      cls
      locate  7,2
      print  dstring$
      system
endif

if (request% = 99) then
       if (retcode% <>   and dlength% <>  ) then
            return
      else
       '   conversion error
            cls
            locate  7,2
             print "function 99 -- error occurred in conversion"
            locate  8,2
             print "received return code" retcode%
            locate  9,2
             print "received length          " dlength%
            system
      endif
endif

'   return code was ok
if ((retcode% <> expectedrc%) and (expectedrc% <> -1)) then
      cls
      locate  7,2
       print "function" request%
      locate  8,2
       print "expected return code" expectedrc%
      locate  9,2
       print "received return code" retcode%
      system
endif

return
```

IBM C Sample Program

Function Name	Function Number
Connect	1
Disconnect	2
Send Key	3
Wait	4
Search Presentation Space	6
Set Session Parameters	9
Copy String to Presentation Space	15
Pause	18
Query System	20
Reset System	21
Start Host Notification	23
Query Host Update	24
Stop Host Notification	25
Send File	90

```
/                                                  /
/                                                                  /
/                          SAMPLE EHLLAPI PROGRAM                          /
/                                                                  /
/      Author:    IBM                                              /
/      Date  Written:    1988                                     /
/       For use with PC/327                  /
/                                                                  /
/       Certain VM/37  keywords used below may vary in function    /
/            depending on system configuration.    A host          /
/            presentation space size of 192  is assumed.          /
/                                                                  /
/                                                  /


#define PSID "a"
#define PS_POS return_code
#define DRV_NUM return_code
#define CLEAR "@C"
#define ENTER "@E"

/  External variables  /

        void  call_hllapi();
        void  send_keys();

         int i;                          /  general counter  /

         int row;                        /  cursor control  /
        int  column;

         int DRV_NUM;                       /  for file transfer  /
         int PS_POS;                      /  for PS search  /
        int  search_rc;
```

```
          int function;                    /  call parameters  /
         char  data_string[3 ];
         int  data_length;
         int  return_code;

          int expected_return_code;      /  check call result  /

         char  sf_string[47]=
                  {"autoexec.bat a:auto bat a (ascii crlf!"};

          char q_buffer[35];               /  for query system  /

main()
{
/                                         /
/   Reset  EHLLAPI.                                                    /
/                                         /
          function = 21;
          expected_return_code =  ;
         call_hllapi();

/                                         /
/    Establish connection to the host presentation space.          /
/                                         /
          function = 1;
          strcpy (data_string, PSID);
          expected_return_code =  ;
         call_hllapi();

/                                         /
/    Send the system and user ids to the host.                      /
/                                         /
          strcpy (data_string, "systemid userid@E");
          data_length = 17;
         send_keys();

/                                         /
/    Has CP asked for my password?    If not, wait for a host       /
/       update and check again.    Try this twice; if it still      /
/       hasn't asked, the system is either down or very slow.     /
/       In this case, just terminate.                                  /
/                                         /
          function = 6;
          strcpy (data_string, "Enter your password");
          data_length = 19;
          expected_return_code = -1;
         call_hllapi();

          if (return_code !=  )
              {
                function = 23;
                strcpy (data_string, "ap        ");
                expected_return_code =  ;
               call_hllapi();

                function = 9;
                strcpy (data_string, "ipause");
                data_length = 6;
```

```
        expected_return_code =  ;
       call_hllapi();

        search_rc = 24;
        i = 1;

        while ((i <= 2) && (search_rc !=  ))
            {
              function = 18;
              data_length = 1 ;
              expected_return_code = -1;
             call_hllapi();

              function = 24;
              strcpy (data_string, PSID);
              expected_return_code = -1;
             call_hllapi();

              function = 6;
              strcpy (data_string, "Enter your password");
              data_length = 19;
              expected_return_code = -1;
             call_hllapi();
              search_rc = return_code;

              i = i + 1;
            }

        if (search_rc !=  )
            {
              printf ("CP never asked for a password - ");
              printf ("sample terminating");
             exit();
            }
      }

/                                                  /
/   Stop host notification and reset 'pause'.                    /
/                                                  /
      function = 25;
      strcpy (data_string, PSID);
      expected_return_code =  ;
     call_hllapi();

      function = 9;
      strcpy (data_string, "fpause");
      data_length = 6;
      expected_return_code =  ;
     call_hllapi();

/                                                  /
/   Send the password & enter.                                    /
/                                                  /
      strcpy (data_string, "password@E");
      data_length = 1 ;
     send_keys();
```

```
/                                                    /
/    Keep clearing the host presentation space until the          /
/       ready prompt (R;) is found.                                    /
/                                                    /
        function = 9;
        strcpy (data_string, "srchfrom");
        data_length = 8;
        expected_return_code =  ;
       call_hllapi();

        search_rc = 24;
        while (search_rc !=  )
            {
              function = 6;
              strcpy (data_string, "MORE");
              data_length = 4;
              PS_POS = 19  ;
              expected_return_code = -1;
             call_hllapi();

              if (return_code ==  )
                  {
                    strcpy (data_string, CLEAR);
                    data_length = 2;
                   send_keys();
                  }

              function = 6;
              strcpy (data_string, "HOLDING");
              data_length = 7;
              PS_POS = 19  ;
              expected_return_code = -1;
             call_hllapi();

              if (return_code ==  )
                  {
                    strcpy (data_string, CLEAR);
                    data_length = 2;
                   send_keys();
                  }

              function = 6;
              strcpy (data_string, "R;");
              data_length = 2;
              PS_POS = 1;
              expected_return_code = -1;
             call_hllapi();

              search_rc = return_code;
            }

        function = 9;
        strcpy (data_string, "srchall");
        data_length = 7;
        expected_return_code =  ;
       call_hllapi();
```

```
/                                        /
/    Set EOT mode on.                                          /
/                                        /
        function = 9;
        strcpy (data_string, "streot eot=!");
        data_length = 12;
        expected_return_code =  ;
       call_hllapi();

/                                        /
/    Block normal messages on VM.                              /
/                                        /
        strcpy (data_string, "set msg off@E!");
       send_keys();

/                                        /
/    Prepare for file transfer by disconnecting from the      /
/       host presentation space.                                /
/                                        /
        function = 2;
        expected_return_code =  ;
       call_hllapi();

/                                        /
/   Upload  "autoexec.bat".                                    /
/                                        /
        function = 9 ;
        DRV_NUM = 4;
        expected_return_code = 3;
        strcopy (data_string, sf_string);
       call_hllapi();

/                                        /
/    Set EOT mode off.                                         /
/                                        /
        function = 9;
        strcpy (data_string, "strlen");
        data_length = 6;
        expected_return_code =  ;
       call_hllapi();

/                                        /
/    Reconnect to the host presentation space.                /
/                                        /
        function = 1;
        strcpy (data_string, PSID);
        expected_return_code =  ;
       call_hllapi();

/                                        /
/    Enable normal messages on VM.                             /
/                                        /
        strcpy (data_string, "set msg on@E");
        data_length = 12;
       send_keys();
```

```
/                                       /
/    Wait for command to complete...                    /
/                                       /
          function = 4;
          expected_return_code =  ;
         call_hllapi();

/                                       /
/    Execute filelist on the host.                      /
/                                       /
          function = 15;
          data_length = 5;
          strcpy (data_string, "filel");
          PS_POS = 1761;
          expected_return_code =  ;
         call_hllapi();

          strcpy (data_string, ENTER);
          data_length = 2;
         send_keys();

/                                       /
/    Wait for command to complete...                    /
/                                       /
          function = 4;
          expected_return_code =  ;
         call_hllapi();

/                                       /
/    Did the file make it to the host?                  /
/                                       /
          function = 9;
          strcpy (data_string, "srchfrom");
          data_length = 8;
          expected_return_code =  ;
         call_hllapi();

          function = 6;
          strcpy (data_string, "AUTO");
          data_length = 4;
          PS_POS = 161;
          expected_return_code =  ;
         call_hllapi();

/                                       /
/   Leave  filelist.                                    /
/                                       /
          strcpy (data_string, "@3");
          data_length = 2;
         send_keys();
         exit();
}


void send_keys()
/                                       /
/    Common sendkey routine.                            /
```

```
/                                                    /
{
          function = 3;
          expected_return_code =  ;
         call_hllapi();
}

void call_hllapi()
/                                                    /
/    Call the hllapi LIM and verify the return code.                 /
/                                                    /
{
          hllc (&function, data_string, &data_length, &return_code);

          if (return_code == 9 && function != 99)
              {
                function = 2 ;
            hllc (&function, q_buffer, &data_length, &return_code);
            row = 7;
            column = 2 ;
            printf ( "[x1B]2J");
            printf ("[x1B]%d;%df", row, column);
           puts  (q_buffer);
           exit();
          }

     if ((return_code != expected_return_code) &&
           (expected_return_code != -1))
         {
           printf ( "[x1B]2J");                          /  Clear the screen  /
           row = 7;
           column = 2 ;
           printf ("[x1B]%d;%df", row, column); /  Set the cursor  /
           printf ("Unexpected return code %d", return_code);
           row = 8;
           printf ("[x1B]%d;%df", row, column); /  Set the cursor  /
           printf ("Function %d", function);
           row = 9;
           printf ("[x1B]%d;%df", row, column); /  Set the cursor  /
           printf ("Expected return code    %d", expected_return_code);
         }
}
```

IBM COBOL Sample Program

Function Name	Function Number
Connect Presentation Space	1
Disconnect Presentation Space	2
Copy Presentation Space	5
Set Session Parameters	9
Copy String to Presentation Space	15
Free All Storage	17
Get Storage	17
Pause	18

IBM COBOL

Function Name	Function Number
Query System	20
Reset System	21
Start Host Notification	23
Query Host Update	24
Stop Host Notification	25
Start Keystroke Intercept	50
Get Key	51
Post Intercept Status	52
Stop Keystroke Intercept	53

```
identification division.
program-id.    sample.
author.    ibm.
date-written.    1988.
remarks.    sample of Cobol/2 EHLLAPI application -- for
                    use with PC/327 .

environment division.
configuration section.
special-names.    console is crt.

data division.
working-storage section.


   A presentation space size of 192  is assumed.    Minor
     Cobol/2 features are used that may not be supported
     by Cobol 2. .

1    1  -hllapi-parameters.
       5    1  -hllapi-function                  pic 9( 4) usage comp- .
       5    1  -hllapi-data-string              pic  x(192 ).
       5    1  -hllapi-data-length              pic 9( 4) usage comp- .
       5    1  -hllapi-return-code              pic 9( 4) usage comp- .
       5    1  -hllapi-ps-position
              redefines  1  -hllapi-return-code
                                                    pic 9( 4) usage comp- .

1    2  -hllapi-buffers.
       5    21 -hllapi-storage-buffer.
              1     21 -hllapi-storage-session       pic x( 2) value 'aB'.
              1     21 -hllapi-storage-address       pic  x( 4).
       5    22 -hllapi-keystroke-buffer.
              1     22 -hllapi-storage-session       pic x( 1) value 'a'.
              1     22 -hllapi-option-code             pic x( 1) value space.
              1     22 -hllapi-keystroke               pic  x( 4).
              1     22 -hllapi-keystroke-pos3-4      pic  x( 2)
                     redefines  22 -hllapi-keystroke.
              1     filler                               pic  x( 2)
                                                            value  low-values.

1    3  -expected-return-code                pic 9( 4) usage comp- .

1    4  -query-system-string.
```

```
      5   4  -qs-hllapi-version              pic  x( 1).
      5   4  -qs-hllapi-level              pic  x( 2).
      5   4  -qs-hllapi-date               pic  x( 6).
      5   filler                             pic  x( 3).
      5   4  -qs-hardware-base             pic  x( 1).
      5   4  -qs-cp-type                    pic  x( 1).
      5   filler                             pic  x( 2).
      5   4  -qs-pscomm-version            pic  x( 2).
      5   filler                             pic  x( 1).
      5   4  -qs-error-code-one            pic  x( 4).
      5   4  -qs-error-code-two            pic  x( 4).
       5   4  -qs-model                       pic 9( 4) usage comp.
       5   4  -qs-nls-type                  pic 9( 4) usage comp.
      5   4  -qs-monitor                    pic  x( 1).
      5   filler                             pic  x( 3).

1   5  -messages.
       5   5  -function-name                 pic x(25) value spaces.
       5   5  -message                        pic x(8 ) value spaces.

screen section.

1   1  -host-screen.
      5   blank  screen.
      5   1  -host-layout.
           1   line   1  column   1  pic  x(192 )
                                    from  1  -hllapi-data-string
                                     background-color is 1
                                     foreground-color is 7.
           1    line 25 column    1 value
                'compare this to host, then press enter to continue'
                                     foreground-color is 3.

1   2  -return-screen.
      5   blank  screen.
      5   2  -return-layout
           foreground-color is 3.
           1   line   3  column   1  value  'function'.
           1   line   3  column  14  pic  z( 3)9
                                    from  1  -hllapi-function.
           1   line   3  column  19  value  '--'.
           1   line   3  column  22  pic  x(25)
                                    from  5  -function-name.
           1   line   4  column   1  value 'return  code'.
           1   line   4  column  13  pic  -z( 3)9
                                    from  1  -hllapi-return-code.
           1   line   5  column   1  pic  x(8 )
                                    from  5  -message.

1   3  -query-system-screen.
      5   blank  screen.
      5   3  -query-system-layout
           foreground-color is 4.
           1   line   6  column   1  value 'Query  System'.
           1   line   8  column   1  value 'EHLLAPI  version  number  -->'.
           1   line   8  column  28  pic  x( 1)
                                    from  4  -qs-hllapi-version.
           1   line   9  column   1  value 'EHLLAPI  level  number  ---->'.
```

```
           1   line  9  column  28  pic  x( 2)
                                    from  4  -qs-hllapi-level.
           1    line 1  column    1 value 'EHLLAPI date ---------->'.
           1    line 1  column 28 pic x( 6)
                                    from  4  -qs-hllapi-date.
           1    line 12 column    1 value '327  P.C. version ------>'.
           1    line 12 column 28 pic x( 2)
                                    from  4  -qs-pscomm-version.
           1    line 14 column    1 value 'error code one --------->'.
           1    line 14 column 28 pic x( 4)
                                    from  4  -qs-error-code-one.
           1    line 15 column    1 value 'error code two --------->'.
           1    line 15 column 28 pic x( 4)
                                    from  4  -qs-error-code-two.


============================================================
procedure division.

  -start.
       perform 1  -reset-system thru 1  -exit.
       perform 11 -connect-pres-space thru 11 -exit.

       perform 2  -copy-str-to-ps thru 2  -exit.
       perform 21 -copy-pres-space thru 21 -exit.

       perform 3  -get-storage thru 3  -exit.
       perform 31 -start-host-notif thru 31 -exit.
       perform 32 -set-ipause-parm thru 32 -exit.
       perform 33 -pause thru 33 -exit.
       perform 34 -query-host-update thru 34 -exit.
       perform 35 -stop-host-notif thru 35 -exit.

       perform 4  -start-keystroke-intercept thru 4  -exit.
       perform 41 -get-keystroke thru 41 -exit.
       perform 42 -post-intercept-status thru 42 -exit.
       perform 43 -stop-keystroke-intercept thru 43 -exit.
       perform 44 -free-all-storage thru 44 -exit.

       perform 5  -disconnect-pres-space thru 5  -exit.
       perform 1  -reset-system thru 1  -exit.
      stop  run.

1  -reset-system.

    function 21    set initial hllapi status

       move   to 3  -expected-return-code.
       move 'reset system' to 5  -function-name.
       move 21 to 1  -hllapi-function.
       perform 6  -call-hllapi thru 6  -exit.
1  -exit.
      exit.

11 -connect-pres-space.

    function 1    connect to host
```

```
       move   to 3  -expected-return-code.
       move 'connect' to 5  -function-name.
       move 1 to 1  -hllapi-function.
       move 'a' to 1  -hllapi-data-string.
       perform 6  -call-hllapi thru 6  -exit.
11 -exit.
      exit.

2  -copy-str-to-ps.

    function 15 copy string to host screen

       move   to 3  -expected-return-code.
       move 'copy string to ps' to 5  -function-name.
       move 15 to 1  -hllapi-function.
       move 'hello host' to 1  -hllapi-data-string.
       move 1  to 1  -hllapi-data-length.
       move 182  to 1  -hllapi-ps-position.
       perform 6  -call-hllapi thru 6  -exit.
2  -exit.
      exit.

21 -copy-pres-space.

    function 5    copy host screen to buffer

       move   to 3  -expected-return-code.
       move 'copy ps' to 5  -function-name.
       move 5 to 1  -hllapi-function.
       perform 6  -call-hllapi thru 6  -exit.
      display  1  -host-screen.
       stop ' '.
21 -exit.
      exit.

3  -get-storage.

    function 17    retrieve address of host notification buffer

       move   to 3  -expected-return-code.
       move 'get storage' to 5  -function-name.
       move 17 to 1  -hllapi-function.
       move 256 to 1  -hllapi-data-length.
       move 1 to 1  -hllapi-ps-position.
       perform 6  -call-hllapi thru 6  -exit.
       move 1  -hllapi-data-string to 21 -hllapi-storage-address.
3  -exit.
      exit.

31 -start-host-notif.

    function 23    begin host monitoring

       move 'Switch to the A host session and type IBM'
            to  5  -message.
       move   to 3  -expected-return-code.
       move 'start host notification' to 5  -function-name.
       move 23 to 1  -hllapi-function.
```

```
        move 256 to 1  -hllapi-data-length.
        move 21 -hllapi-storage-buffer to 1  -hllapi-data-string.
        perform 6  -call-hllapi thru 6  -exit.
31 -exit.
       exit.

32 -set-ipause-parm.

   function   9   use  interruptible  pause

        move   to 3  -expected-return-code.
        move 'set parm - ipause' to 5  -function-name.
        move 9 to 1  -hllapi-function.
        move 'IPAUSE' to 1  -hllapi-data-string.
        move 6 to 1  -hllapi-data-length.
        perform 6  -call-hllapi thru 6  -exit.
32 -exit.
       exit.

33 -pause.

    function 18    wait up to 1  seconds for a host update

        move 26 to 3  -expected-return-code.
        move 'pause' to 5  -function-name.
        move 18 to 1  -hllapi-function.
        move 2  to 1  -hllapi-data-length.
        perform 6  -call-hllapi thru 6  -exit.
33 -exit.
       exit.

34 -query-host-update.

    function 24    has a host update occurred?

        move 23 to 3  -expected-return-code.
        move 'query host update' to 5  -function-name.
        move 24 to 1  -hllapi-function.
        move 'a' to 1  -hllapi-data-string.
        perform 6  -call-hllapi thru 6  -exit.
34 -exit.
       exit.

35 -stop-host-notif.

    function 25    end host monitoring

        move   to 3  -expected-return-code.
        move 'stop host notification' to 5  -function-name.
        move 25 to 1  -hllapi-function.
        move 'a' to 1  -hllapi-data-string.
        perform 6  -call-hllapi thru 6  -exit.
35 -exit.
       exit.

4  -start-keystroke-intercept.
```

```
    function 5     begin keyboard monitoring

        move 'Press the host enter key!    Then return and press enter
-       'to continue.' to 5  -message.
        move   to 3  -expected-return-code.
        move 'start keystroke intercept' to 5  -function-name.
        move 5  to 1  -hllapi-function.
        move 'aL' to 21 -hllapi-storage-session.
        move 256 to 1  -hllapi-data-length.
        move 21 -hllapi-storage-buffer to 1  -hllapi-data-string.
        perform 6  -call-hllapi thru 6  -exit.
 4  -exit.
       exit.

 41 -get-keystroke.

    function 51    retrieve keystroke from user

        move   to 3  -expected-return-code.
        move 'get keystroke' to 5  -function-name.
        move 51 to 1  -hllapi-function.
        move low-values to 22 -hllapi-keystroke.
        move 22 -hllapi-keystroke-buffer to 1  -hllapi-data-string.
        perform 6  -call-hllapi thru 6  -exit.
 41 -exit.
       exit.

 42 -post-intercept-status.

    function 52    correct keystroke or not?

        move 1  -hllapi-data-string to 22 -hllapi-keystroke-buffer.
        if (22 -hllapi-option-code = 'M') and
             (22 -hllapi-keystroke-pos3-4 = '@E')
               move 'You pressed the ENTER key!' to 5  -message
               move 'aA' to 1  -hllapi-data-string
       else
               move 'You did not press the ENTER key!' to 5  -message
               move 'aR' to 1  -hllapi-data-string.

        move   to 3  -expected-return-code.
        move 'post intercept status' to 5  -function-name.
        move 52 to 1  -hllapi-function.
        perform 6  -call-hllapi thru 6  -exit.
 42 -exit.
       exit.
 43 -stop-keystroke-intercept.

    function 53    end keyboard monitoring

        move   to 3  -expected-return-code.
        move 'stop keystroke intercept' to 5  -function-name.
        move 53 to 1  -hllapi-function.
        move 'a' to 1  -hllapi-data-string.
        perform 6  -call-hllapi thru 6  -exit.
 43 -exit.
       exit.
```

```
44 -free-all-storage.

     function 17    release host/keyboard notification buffer.

         move   to 3  -expected-return-code.
         move 'free all storage' to 5  -function-name.
         move 17 to 1  -hllapi-function.
         move 4 to 1  -hllapi-return-code.
         perform 6  -call-hllapi thru 6  -exit.
44 -exit.
       exit.

5  -disconnect-pres-space.

    function   2   disconnect  from  host

         move   to 3  -expected-return-code.
         move 'disconnect' to 5  -function-name.
         move 2 to 1  -hllapi-function.
         perform 6  -call-hllapi thru 6  -exit.
5  -exit.
       exit.

51 -query-system.

     function 2     retrieve system information

         move   to 3  -expected-return-code.
         move 2  to 1  -hllapi-function.
         call 'hllcob' using
               1  -hllapi-function
               4  -query-system-string
               1  -hllapi-data-length
               1  -hllapi-return-code.
       display  3  -query-system-screen.
       stop  run.
51 -exit.
       exit.

6  -call-hllapi.

     execute the designated hllapi function and check the
       return  code.

         call 'hllcob' using
               1  -hllapi-function
               1  -hllapi-data-string
               1  -hllapi-data-length
               1  -hllapi-return-code.

       display  2  -return-screen.

         if (1  -hllapi-return-code not = 3  -expected-return-code)
              if (1  -hllapi-return-code <  )
                   display '    emulator not started - application termin
-       'ated    '      at line 15 column 13
                              with bell highlight foreground-color 4
                   stop  run
```

```
           else
                if (1  -hllapi-return-code = 9)
                     perform 51 -query-system thru 51 -exit
                else
                     display 'unexpected return code'
                          at line 5 column 1.

          stop ' '.
          move spaces to 5  -function-name.
          move spaces to 5  -message.
 6  -exit.
         exit.
```

IBM Pascal Compiler Sample Program

Function Name	Function Number
Connect	1
Disconnect	2
Send Key	3
Wait	4
Search Presentation Space	6
Copy Presentation Space to String	8
Reserve	11
Release	12
Query System	20
Reset System	21

```
program set_time (input, output);

( ============================================================ )
(                                                              )
(                        SAMPLE EHLLAPI PROGRAM                )
(                                                              )
(    The time and date are obtained from the host session and  )
(       passed to the PC session.                              )
(                                                              )
(    Author:   IBM                                             )
(    Date  Written:   1988                                     )
(    For use with PC/327                                       )
(                                                              )
( ============================================================ )


    ( ========================================================== )
    (                                                          )
     (    Keyboard mnemonics -- these are valid only when the  )
     (       default escape character '@' is not changed by    )
     (       function 9 (set session parameters).              )
    (                                                          )
    ( ========================================================== )
const
    clear_key  =  '@C';
    enter_key  =  '@E';
```

```
    reset_key  =  '@R';


   ( ========================================================== )
   (                                                            )
    (   Date and time record types for passing information to    )
    (      the PC session.                                       )
   (                                                            )
   ( ========================================================== )
type
    date_record = record
        year,
        month,
        day         :  integer;
   end;

    time_record = record
        hour,
        min,
        sec,
        tick        :  integer;
   end;


var
   ( ========================================================== )
   (                                                            )
    (   EHLLAPI calling parameters.                             )
   (                                                            )
   ( ========================================================== )
   hllapi_function        :  integer;
    hllapi_data_string    : string (192 );
   hllapi_data_length     :  integer;
   hllapi_return_code     :  integer;


   ( ========================================================== )
   (                                                            )
    (   Time and date information.                              )
   (                                                            )
   ( ========================================================== )
    time_string,                            ( time : hh:mm:ss  )
    date_string : string(8);                ( date : mm/dd/yy  )

    time : time_record;
    date : date_record;


   ( ========================================================== )
   (                                                            )
    (   External references to Pascal/2 library functions for   )
    (      setting the time and date and for terminating the     )
    (      program.   See the IBM Pascal Compiler/2 Language     )
    (      Reference for details.                                )
   (                                                            )
   ( ========================================================== )
    function setdat (year, month, day : integer)
        : boolean; EXTERN;
```

```
 function settim (hours, minutes, seconds, hundreds : integer)
      : boolean; EXTERN;

 procedure endxqq; extern;


( ========================================================== )
(                                                            )
 (    External reference to EHLLAPI Pascal LIM.              )
(                                                            )
( ========================================================== )
 procedure hllpas (var lim_func          : integer;
                       var  lim_string      :  string;
                       var  lim_length,
                            lim_rc          : integer); extern;


 function digit (ch : char) : integer;
( ========================================================== )
(                                                            )
 (    Converts a digit in character format to integer format.  )
(                                                            )
( ========================================================== )
begin
      digit := ord(ch) - ord(' ')
end;

procedure  call_hllapi;
( ========================================================== )
(                                                            )
 (    Call HLLPAS and check the return code.                 )
(                                                            )
( ========================================================== )
type
      qsysinfo = record
      qs_hllvers [  ] : string(1);
      qs_hlllvl    [ 1] : string(2);
      qs_hlldate [ 3] : string(6);
      qs_rsrv1      [ 9] : string(3);
      qs_hardw      [12] : string(1);
      qs_cptype    [13] : string(1);
      qs_rsrv2      [14] : string(2);
      qs_pcvers    [16] : string(2);
      qs_rsrv3      [18] : string(1);
      qs_syserr1 [19] : string(4);
      qs_syserr2 [23] : string(4);
      qs_model      [27] : string(2);
      qs_rsrv4      [29] : string(2);
      qs_monitor [31] : string(1);
      qs_rsrv5      [32] : string(3);
end;

var
      q_sys_str : string(35);
      q_sys_ptr : ADS of qsysinfo;

begin
```

```
        hllpas (hllapi_function, hllapi_data_string,
                  hllapi_data_length,  hllapi_return_code);

        if (hllapi_return_code = 9) then
            begin
                hllapi_function := 2 ;
                hllapi_data_length := 35;
                hllpas (hllapi_function, hllapi_data_string,
                           hllapi_data_length,  hllapi_return_code);

                mover (ADR(hllapi_data_string), ADR(q_sys_str), 35);


           q_sys_ptr := ADS q_sys_str;
           with q_sys_ptr[ do
               begin
                   writeln('  ');
                   writeln('Query  System');
                   writeln('  ');
                    writeln('EHLLAPI version number ----> ', qs_hllvers);
                    writeln('EHLLAPI level number ------> ', qs_hlllvl);
                    writeln('EHLLAPI date --------------> ', qs_hlldate);
                    writeln('327  PC version ----------> ', qs_pcvers);
                    writeln('hardware base ------------> ', qs_hardw);
                    writeln('model type ---------------> ', qs_model);
                    writeln('monitor type -------------> ', qs_monitor);
                    writeln('control program level ----> ', qs_cptype);
                    writeln('system error code one ----> ', qs_syserr1);
                    writeln('system error code two ----> ', qs_syserr2);
                   writeln('  ')
               end;
          endxqq
      end
end;


procedure convert_info;
( ============================================================ )
(                                                              )
(    Converts the time and date strings to integer format      )
(       for use in 'settim' and 'setdat' function calls.       )
(                                                              )
( ============================================================ )
begin
 time.hour    := 1    digit(time_string[1]) + digit(time_string[2]);
 time.min     := 1    digit(time_string[4]) + digit(time_string[5]);
 time.sec     := 1    digit(time_string[7]) + digit(time_string[8]);
 time.tick    :=  ;
 date.month := 1    digit(date_string[1]) + digit(date_string[2]);
 date.day     := 1    digit(date_string[4]) + digit(date_string[5]);
 date.year    := 1    digit(date_string[7]) + digit(date_string[8])
                          +  19
end;


     procedure get_info (const delimiter : lstring);
    ( ========================================================== )
    (                                                            )
```

```
 (    Searches the host screen for the given delimiter.    If       )
 (        the delimiter is found, then the string of                  )
 (        information (time or date) is copied into the given       )
(        string.                                                      )
( ============================================================ )
var
     delimiter_position   :   integer;

begin
     (  Search Presentation Space  )
     hllapi_function := 6;
     copystr (delimiter, hllapi_data_string);
     hllapi_data_length := 1;
    call_hllapi;

     (  Length >   indicates the string was found  )
     delimiter_position := hllapi_data_length;
     if (delimiter_position >  ) then
         begin
              hllapi_function := 8;
              hllapi_data_length := 8;
              hllapi_return_code := delimiter_position - 2;
             call_hllapi
         end
end;


procedure  query_for_info;
( ============================================================ )
(                                                                )
 (    Attempt to query the host for the time and date.            )
(                                                                )
( ============================================================ )
var
     cursor_location   :   integer;

begin
     (  === Clear host screen ===  )
     hllapi_function := 3;
     copystr (clear_key, hllapi_data_string);
     hllapi_data_length := 2;
    call_hllapi;

     (  === Wait for ready host ===  )
     hllapi_function := 4;
    call_hllapi;

     (  === Search presentation space ===  )
     hllapi_function := 6;
     copystr ('RUNNING', hllapi_data_string);
     hllapi_data_length := 7;
    call_hllapi;

     (  === Query host for time ===  )
     hllapi_function := 3;
     copystr ('query time' enter_key, hllapi_data_string);
     hllapi_data_length := 12;
    call_hllapi;
```

```
      (  === Wait for host response ===  )
      hllapi_function := 4;
     call_hllapi;

     get_info  (':');
      mover (ADR(hllapi_data_string), ADR(time_string), 8);
     get_info  ('/');
      mover (ADR(hllapi_data_string), ADR(date_string), 8)
end;


procedure  get_time_from_host;
( ========================================================== )
(                                                            )
 (    The time and date are retrieved while the host session     )
 (       is being controlled.                                    )
(                                                            )
( ========================================================== )
begin
     (  === Connect to the host session ===  )
     hllapi_function := 1;
     copystr ('a', hllapi_data_string);
     call_hllapi;

     (  === Reserve the host keyboard ===  )
     hllapi_function := 11;
     call_hllapi;

     query_for_info;

     (  === Release the host keyboard ===  )
     hllapi_function := 12;
     call_hllapi;

     (  === Disconnect from the host session ===  )
     hllapi_function := 2;
     call_hllapi
end;


procedure  give_time_to_pc;
( ========================================================== )
(                                                            )
 (    The time and date are converted to integer format and      )
 (       passed to the Pascal/2 functions "setdat" and             )
 (       "settim".    If either of these functions fail, a          )
 (       warning is issued.                                      )
(                                                            )
( ========================================================== )
begin
     convert_info;

     if not settim(time.hour, time.min, time.sec, time.tick) then
          writeln ('Time was not set');

     if not setdat(date.year, date.month, date.day) then
          writeln ('Date was not set')
```

```
end;


(                                        )
(                                  MAIN                                  )
(                                                                        )
 (     The EHLLAPI session is reset before and after the          )
 (        core of the application.                                  )
(                                                                        )
(                                        )

begin
     hllapi_function := 21;
    call_hllapi;

    get_time_from_host;

    give_time_to_pc;

     hllapi_function := 21;
    call_hllapi;
end.
```

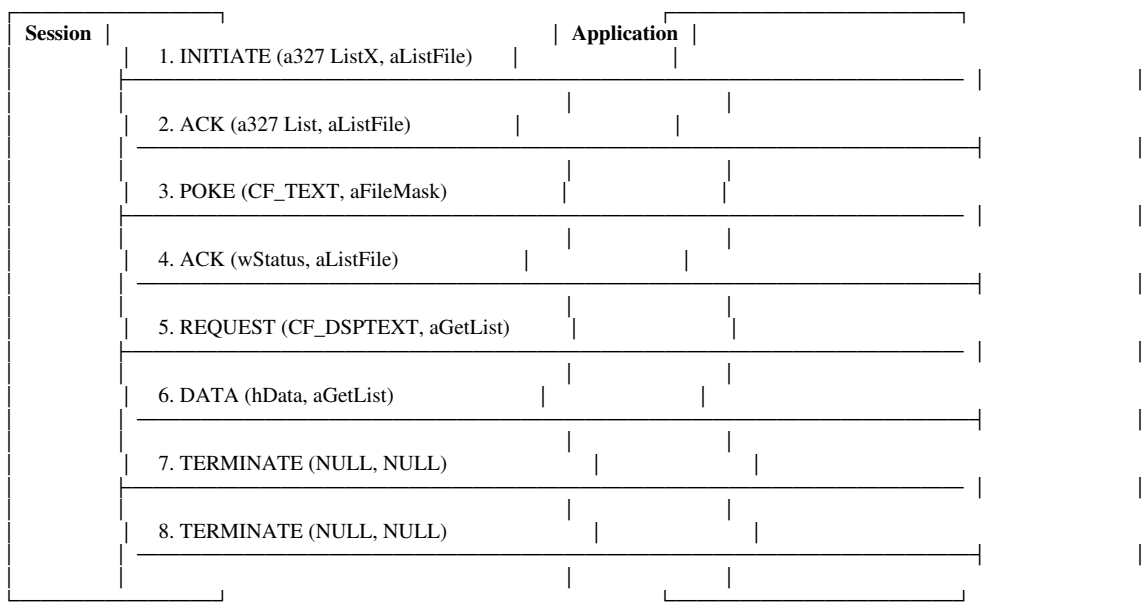
IBM Pascal Compiler

Appendix D. DDE Sample Program

Creating a DDE Program to Support the Receive Load Button

PC/3270 or PC400 Windows file transfer receive function allows you to create a DDE application that will provide the receive dialog with a list of host files to down load. The DDE program can connect to the session and issue host commands to retrieve a list of file names from the host, or generate the list in any other automated fashion.

The following message flow takes place when the load button is pressed:



If an error occurs or an unexpected DDE message is received by the application, the application is expected to recover from the error and send a TERMINATE message to the PC/3270 or PC400 session. If PC/3270 or PC400 Windows session encounters an error, it will send a TERMINATE message to the application to terminate the conversation.

The following program, LISTFILE, is an example of a DDE application that supports the load button. The following files comprise the LISTFILE application:

LISTFILE Make file for the LISTFILE application

LISTFILE.C C source file containing the window procedure

LISTDDE.C C source file containing the DDE handler

LISTFILE.H C header file that defines structures

LISTDATA.H C header file that contains data declarations

LISTFILE.ICO Icon file for the LISTFILE application

LISTFILE.RC Resource file for the LISTFILE application

LISTFILE.DEF Definition file for the LISTFILE application

© Copyright IBM Corp. 1996

To run the program, invoke it as a Windows application. When the user presses the load button in the receive dialog, the conversation will begin between LISTFILE and the PC/3270 or PC400 Windows session.

The following sample program files are on the *Personal Communications/3270 for Windows* diskette. The sample programs in this publication might differ from the programs on the diskette. The diskette contains the current sample programs.

LISTFILE.C File

```
//
//
//   PROGRAM:  listfile.c
//
//    PURPOSE: Provides, via DDE, a list of host files to the PC/327  receive
//                  files  function.
//
//   1.  Each  instance  of  this  application  corresponds  to  one  session
//           instance. This allows load button enablement on a session by session
//          basis.
//
//   2.  Invocation:  LISTFILE  /x     where  "x"  is  the  alpha  session  ID.  A  ..  H
//
//                Copyright (c) 1991, IBM Corporation. All rights reserved.
//
//                Provided on an "AS IS" basis, no warranty expressed or implied.
//
//


#include "windows.h"                        // required for all Windows applications
#include "dde.h"                            // required for all Windows DDE Appls
                                            //
#include <string.h>                         // String functions
#include <ctype.h>                          // toupper function
                                            //
#include "listfile.h"                       // specific to this program

#define MAIN
#include "listdata.h"                       // Global Data
                                            //
//
//
// Local Function Definitions
//
//

BOOL NEAR GetSessionID( LPSTR );
```

Note:

FUNCTION: WinMain(HANDLE, HANDLE, LPSTR, int)

PURPOSE: calls initialization functions, and processes the application
message loop.

COMMENTS:

Windows recognizes this function by name as the initial entry point
for the program. This function calls the application initialization
routine, if no other instance of the program is running, and always
calls the instance initialization routine. It then executes a message
retrieval and dispatch loop that is the top-level control structure
for the remainder of execution. The loop is terminated when a WM_QUIT
message is received, at which time this function exits the application
instance by returning the value passed by PostQuitMessage().

```
int PASCAL WinMain(HANDLE hInstance, HANDLE hPrevInstance, LPSTR lpzCmdLine, int
 nCmdShow)
{
  MSG  msg;

   if ( !GetSessionID(lpzCmdLine) )                    // Get Session ID A .. H
   {                                                        //
      LoadString(hInstance, LS_COMMAND_LINE_ERROR1, zMsgBuffer, sizeof(zMsgBuffer)
);
      MessageBox(NULL, (LPSTR) zMsgBuffer, (LPSTR) ListFileClassName, MB_OK | MB_I
CONQUESTION );
      return( FALSE );                                  //
   }                                                        //
                                                            //
  if  (!hPrevInstance)                                  //
   {                                                        //
     if  (!RegisterApplicationClass(hInstance))    //
     {                                                      //
        return( FALSE );                                //
     }                                                      //
   }                                                        //
                                                            //
   if (!InitInstance(hInstance, nCmdShow))          //
   {                                                        //
      return( FALSE );                                  //
   }                                                        //
                                                            //
   if ( !InitDDE() )                                    //
   {                                                        //
      return( FALSE );                                  //
   }                                                        //
                                                            //
   //    Dispatch messages until a WM_QUIT message is received.
                                                            //
```

```
   while( GetMessage(&msg, NULL, NULL, NULL) )    //
  {                                                //
      TranslateMessage(&msg);                     // Translates virtual key codes
      DispatchMessage(&msg);                       // Dispatches message to window
  }                                                //
                                                    // Return the PostQuitMessage()
  return  (msg.wParam);                            //  Value
}                                                  //
                                                   //
                                                   //
```

Note:

FUNCTION: RegisterApplicationClass(HANDLE)

PURPOSE: Initializes window data and registers window class

This function is called at initialization time only if no other instances of the application are running. This function performs initialization tasks that can be done once for any number of running instances.

In this case, we initialize a window class by filling out a data structure of type WNDCLASS and calling the Windows RegisterClass() function. Since all instances of this application use the same window class, we only need to do this when the first instance is initialized.

```
BOOL RegisterApplicationClass( HANDLE hInstance )
{
  WNDCLASS   wc;

   // Fill in window class structure with parameters that describe the
   // main window.

  wc.hInstance         =  hInstance;
   wc.hIcon              = LoadIcon(hInstance, MAKEINTRESOURCE(LISTICON));
   wc.hCursor            = LoadCursor(NULL, IDC_ARROW);
   wc.hbrBackground = GetStockObject(LTGRAY_BRUSH);
  wc.cbClsExtra       =  NULL;
  wc.cbWndExtra       =  NULL;
   wc.style              = CS_VREDRAW | CS_HREDRAW | CS_DBLCLKS;
  wc.lpszMenuName   =  MAKEINTRESOURCE(LISTMENU);
   wc.lpszClassName = ListFileClassName;
  wc.lpfnWndProc     =  ListFileWndProc;

   // Register the window class and return FALSE if the call fails

   if( !RegisterClass(&wc) )
   {
     return(FALSE);
   }

   // Fill in window class structure with parameters that describe the
   // hidden window used to converse with session to obtain the file name
  //  list.
```

```
  wc.hInstance     = hInstance;
  wc.hIcon         = NULL;
  wc.hCursor       = LoadCursor(NULL, IDC_ARROW);
  wc.hbrBackground = GetStockObject(LTGRAY_BRUSH);
  wc.cbClsExtra    = NULL;
  wc.cbWndExtra    = NULL;
  wc.style         = CS_VREDRAW | CS_HREDRAW;
  wc.lpszMenuName  = NULL;
  wc.lpszClassName = FileListClassName;
  wc.lpfnWndProc   = FileListWndProc;

  // Register the window class and return FALSE if the call fails

  return( RegisterClass(&wc) );
}
```

Note:

FUNCTION: InitInstance(HANDLE, int)

PURPOSE: Saves instance handle and creates main window

This function is called at initialization time for every instance of this application. This function performs initialization tasks that are unique to this instance.

In this case, we save the instance handle in a static variable and create and display the main program window.

```
BOOL InitInstance(HANDLE hInstance, int iCmdShow)
{
  HWND  hWnd;
  int iX, iY;

  // Save the instance handle in static variable, which will be used in
  // many subsequence calls from this application to Windows.

  hGlobalInst = hInstance;

  // Create a main window for this application instance.

  iX = GetSystemMetrics( SM_CXSCREEN );
  iY = GetSystemMetrics( SM_CYSCREEN );

  LoadString(hInstance, LS_TITLE_TEXT, zMsgBuffer, sizeof(zMsgBuffer));

  strcat(zMsgBuffer,  zCmdLineSessionID);

  hWnd = CreateWindow(
      ListFileClassName,                  // See RegisterClass() call.
      zMsgBuffer,                         // Text for window title bar.
      WS_OVERLAPPEDWINDOW,                // Window style.
      (iX / 4),                           // Horizontal position.
      (iY / 4),                           // Vertical position.
      (iX / 2),                           // Width
      (iY / 2),                           // Height
      NULL,                               // Overlapped windows have no parent.
```

```
        NULL,                                   // Use the window class menu.
        hInstance,                             // This instance owns this window.
        NULL                                    // Pointer not needed.
  );

  // If window could not be created, return FALSE

  if ( hWnd == NULL )
  {
     return(FALSE);
  }

  hGlobalWnd = hWnd;                           // Save Window Handle

  // Make the window visible; update its client area; and return "success"

  ShowWindow(hWnd, SW_MINIMIZE);              // Show the window
  UpdateWindow(hWnd);                          // Sends WM_PAINT message
                                               //
  // Create the Hidden window                 //
                                               //
  hWndHidden = CreateWindow(                   //
        FileListClassName,                    // See RegisterClass() call.
        "",                                    // Text for window title bar.
        WS_OVERLAPPEDWINDOW,                        // Window style.
        (iX / 4),                             // Horizontal position.
        (iY / 4),                             // Vertical position.
        (iX / 2),                             // Width
        (iY / 2),                             // Height
        NULL,                                    // Overlapped windows have no parent.
        NULL,                                    // Use the window class menu.
        hInstance,                             // This instance owns this window.
        NULL                                     // Pointer not needed.
  );                                           //
                                               //
  // If window could not be created, return FALSE
                                               //
  if ( hWndHidden == NULL )                    //
  {                                            //
     return(FALSE);                            //
  }                                            //
                                               //
  return(TRUE);                                // Return success
}
```

Note:

FUNCTION: ListFileWndProc(HWND, unsigned, WORD, LONG)

PURPOSE: Main Window Procedure for LISTFILE application

This procedure handles most all of the message traffic for the
application. When this procedure receive a WM_QUIT message the application
will terminate by exiting the while message loop in the function WinMain.

```
long FAR PASCAL ListFileWndProc(HWND hWnd, unsigned message, WORD wParam, LONG l
Param)
{
  FARPROC          lpProcModalDialog;
  PAINTSTRUCT   ps;

  switch  (message)
  {
     case  WM_COMMAND:
         switch( wParam )
        {
           case  ID_ABOUT:
               lpProcModalDialog = MakeProcInstance(About, hGlobalInst);
               DialogBox(hGlobalInst, MAKEINTRESOURCE(ABOUTBOX), hWnd, lpProcModalDia
log);
              FreeProcInstance(lpProcModalDialog);
              break;

           default:
               return( DefWindowProc(hWnd, message, wParam, lParam) );
              break;
        }
        break;

      case WM_PAINT:      // Paint a Message in the client area
         BeginPaint(hWnd, (LPPAINTSTRUCT) &ps );
         strcpy(zMsgBuffer, "ListFile Sample Application");
         TextOut(ps.hdc, 1 , 1  , (LPSTR) zMsgBuffer, strlen(zMsgBuffer) );
         strcpy(zMsgBuffer, "Copyright (c) 199 , 1991 IBM Corporation.");
         TextOut(ps.hdc, 1 , 12 , (LPSTR) zMsgBuffer, strlen(zMsgBuffer) );
         EndPaint(hWnd, (LPPAINTSTRUCT) &ps );
        break;

      case WM_DESTROY:                          // Destroy the Window
         if( hDDEClient != NULL )              //
         {                                        //
            PostMessage(hDDEClient, WM_DDE_TERMINATE, hWnd,  L );
         }                                        //
        PostQuitMessage( );                    //
        break;                                    //

      //
      //
       //    DDE Messages .....
      //
       //

     case  WM_DDE_INITIATE:
         ListDDEInitiate(hWnd, message, wParam, lParam);
        break;

     case  WM_DDE_POKE:
         ListDDEPoke(hWnd, message, wParam, lParam);
        break;

     case  WM_DDE_REQUEST:
         ListDDERequest(hWnd, message, wParam, lParam);
        break;
```

```
      case  WM_DDE_TERMINATE:
          ListDDETerminate(hWnd, message, wParam, lParam);
          break;

       default:                                    // Pass on all unprocessed messages
          return( DefWindowProc(hWnd, message, wParam, lParam) );
          break;
   }
   return( FALSE );
}
```

Note:

FUNCTION: FileListWndProc(HWND, unsigned, WORD, LONG)

PURPOSE: Window Procedure for the LISTFILE to Session DDE conversation

This window is used to manage the DDE conversation between LISTFILE
and the PC/327 session where the CMS LISTFILE command is issued and the
resulting host file list is obtained.

```
long FAR PASCAL FileListWndProc(HWND hWnd, unsigned message, WORD wParam, LONG l
Param)
{
  switch  (message)
  {
      //
      //
       //   DDE Messages .....
      //
       //

      case  WM_DDE_ACK:
          FileDDEAck(hWnd, message, wParam, lParam);
          break;

      case  WM_DDE_DATA:
          FileDDEData(hWnd, message, wParam, lParam);
          break;

      case  WM_DDE_TERMINATE:
          FileDDETerminate(hWnd, message, wParam, lParam);
          break;

      default:
          return( DefWindowProc(hWnd, message, wParam, lParam) );
          break;
   }
   return( FALSE );
}
```

```
//
//
//   FUNCTION: About(HWND, unsigned, WORD, LONG)
//
//  PURPOSE:  Dialog procedure for the "About" dialog box
//
//  COMMENTS:
//
//        No initialization is needed for this particular dialog box, but TRUE
//        must be returned to Windows.
//
//        Wait for user to click on "Ok" button, then close the dialog box.
//
//

BOOL FAR PASCAL About(HWND hDlg, unsigned message, WORD wParam, LONG lParam)
{
   switch( message )                    // case on the message ID
  {                                       //
      case WM_INITDIALOG:                     // Init dialog
        return(TRUE);                     //
        break;                            //
                                            //
     case  WM_COMMAND:                          //
         switch( wParam )                 //
        {                                   //
           case  IDOK:                      //
           case  IDCANCEL:                    //
              EndDialog(hDlg,  TRUE);       //
              return(TRUE);                 //
              break;                        //
                                            //
           default:                         //
              break;                        //
        }                                   //
        break;                              //
  }                                         //
  return(FALSE);                            //
}

//
//
//   GetSessionID - Parse the short session name from the incoming command
//                          line
//
//   The command line for a session instance will be
//
//        session    /x              x = session ID
//
//

BOOL NEAR GetSessionID( LPSTR lpzCmdLine )
{
   int i, iLength = lstrlen(lpzCmdLine);

   for( i =  ; i < iLength; i++ )
```

```
  {
     if(  lpzCmdLine++ == '/' )
     {
        uchar cID =  lpzCmdLine;                              //

        toupper( cID );

        if( 'A' <= cID && cID <= 'H' )
        {
           zCmdLineSessionID[ ] = (uchar) cID;              // setup string
           zCmdLineSessionID[1] = NULL;                        // Null Terminator
          return(TRUE);
        }
     }
  }
  return(FALSE);
}
```

LISTDDE.C File

Note:

PROGRAM: listdde.c

PURPOSE: DDE interface functions for ListFile DDE application

FUNCTIONS:

ListDDEInitiate - Handle INITIATE for primary DDE conversation
ListDDEAck - Handle ACK for primary DDE conversation
ListDDEPoke - Handle POKE for primary DDE conversation
ListDDERequest - Handle REQUEST for primary DDE conversation
ListDDETerminate - Handle TERMINATE for primary DDE conversation

StartFileConversation - Initiate secondary DDE conversation

FileDDEAck - Handle ACK for secondary DDE conversation
FileDDEData - Handle DATA for secondary DDE conversation
FileDDETerminate - Handle TERMINATE for secondary DDE conversation

lstrncpy - String copy with long pointers
ParsePS - Parse session presentation space (PS)

Copyright (c) 199 , IBM Corporation. All rights reserved.

Note:

In the LISTFILE application two DDE conversations take place. The
primary conversation is initiated by the PC/327 receive dialog function
acting as a DDE client. The LISTFILE application completes the conversation
link by acknowledging the he initiate. from the PC/327 receive dialog.
After LISTFILE has received the file mask to use with the CMS LISTFILE
via a DDE_POKE message, from the client, and received a request for the
the list of files from the client via a DDE_REQUEST message it opens
a hidden window to conduct the secondary conversation. The hidden
window, acting as a client, sends a DDE_INITIATE message to the PC/327
session to start the secondary conversation. Once the secondary
conversation is acknowledged by the PC/327 session the secondary
conversation will send the session the CMS LISTFILE command, and begin
reading the presentation space to retrieve the CMS file names returned
by the CMS listfile command.

```
#include "windows.h"                    // required for all Windows applications
#include "dde.h"                        // required for all Windows DDE Appls
#include <string.h>                     // C string functions
#include <stdio.h>                      // C functions
#include <stdlib.h>                     // C functions
                                        //
#include "listfile.h"                   // specific to this program
#include "listdata.h"                   // Global Data
                                        //
//
//
// Local Function Prototypes
```

```
//
//

BOOL    StartFileConversation (void);                   //Starts 2nd conversation
LPSTR lstrncpy                  (LPSTR,LPSTR,WORD); //copy n chars
WORD    ParsePS                     (LPSTR,WORD,WORD);    //parse presentation space

//
//
// Local Data
//
//

uchar   z327 List[17];
uchar   zGetList[17];
uchar   zFileMask[17];
uchar   zListFile[17];
uchar   zIBM327 [9];
uchar   zSESSION[9];
uchar   zKEYS[6];
uchar   zPS[4];

//
//
//   FUNCTION:  InitDDE()
//
//   PURPOSE:
//
//    Initialize various DDE atoms, and any other required DDE initializations.
//
//

BOOL FAR InitDDE( void )
{
   LoadString(hGlobalInst, LS_16STR_FORMAT, zFmtBuffer, sizeof(zFmtBuffer));
  LoadString(hGlobalInst,  LS_DDE_TOKENS,      zMsgBuffer,  sizeof(zMsgBuffer));

   sscanf(zMsgBuffer,                                   // Input String
             zFmtBuffer,                                  // sscanf format to use
             z327 List,                                   // Server Name
             zListFile,                                   // Server Topic
             zFileMask,                                   // Poke Data Item    file mask
             zGetList,                                    // Request Item for list of files
             zIBM327 ,                                    // Session Server Name
             zSESSION,                                    // Session Topic
             zKEYS,                                       // Session Data Items
            zPS);                                         //  "
                                                          //
   lstrcat(z327 List, zCmdLineSessionID );          // Append Session ID
   lstrcat(zSESSION, zCmdLineSessionID );           // Append Session ID

  return(TRUE);
}
```

Note:

FUNCTION: ListDDEInitiate(HWND , unsigned , WORD , LONG)

PURPOSE:

Accepts the DDE_INITIATE message from the PC/327 receive dialog, and
acknowledges the conversation if it can be supported.

```
void FAR ListDDEInitiate(HWND hWnd, unsigned uMessage, WORD wParam, LONG lParam)
{
   ATOM aApplication    = LOWORD( lParam );
   ATOM aTopic          = HIWORD( lParam );
   ATOM a327 List       = GlobalAddAtom( z327 List );    // Test Atoms
   ATOM aListFile       = GlobalAddAtom( zListFile );    //

   if( aApplication == a327 List )
   {
      if( aTopic == aListFile || aTopic == NULL )
      {
         hDDEClient     = wParam;       // Save away client window handle
         bFirstScreen = TRUE;           // Init first screen flag
         if(!SendMessage(wParam, WM_DDE_ACK, hWnd, MAKELONG(aApplication, aTopic)))
         {
            GlobalDeleteAtom( aApplication ); // Error, delete client's ATOMS
            GlobalDeleteAtom( aTopic       ); // Error, delete client's ATOMS
         }
      }
   }

   GlobalDeleteAtom( a327 List );          // Delete Test Atoms
   GlobalDeleteAtom( aListFile );
}
```

Note:

FUNCTION: ListDDEPoke(HWND , unsigned , WORD , LONG)

PURPOSE:

Processes and acknowledges a DDE_POKE message containing a host file name
mask to be used with the CMS LISTFILE command sent to the VM session.

```
void FAR ListDDEPoke(HWND hWnd, unsigned uMessage, WORD wParam, LONG lParam)
{
   HWND         hWndClient = wParam;                              //
  HANDLE       hFileMask   =  LOWORD(lParam);                      //
   ATOM         aItem       =  HIWORD(lParam);                    //
   ATOM         aFileMask    = GlobalAddAtom( zFileMask ); // Test Atom
   WORD         uAck          = DDE_NACK;                          // Initial ACK Value
  LPGETLIST  lpGetList;                                          //
                                                                  //
   if( aItem == aFileMask && hFileMask != NULL)          //
   {                                                              //
      lpGetList = (LPGETLIST) GlobalLock(hFileMask);     //
      if (lpGetList != NULL)                                 //
      {                                                          //
```

```
        uAck = DDE_ACK;                                          // ACK
        lstrcpy(zHostFileMask,lpGetList->Data);            // Save mask for REQUEST
        if(lpGetList->fRelease)                              // Release the memory
        {                                                          //
          GlobalUnlock(hFileMask);                          //
          GlobalFree(hFileMask);                            //
        }                                                          //
        else                                                      //
        {                                                          //
          GlobalUnlock(hFileMask);                          //
        }                                                          //
      }                                                            //
    }                                                              //
                                                                   //
    if( !PostMessage(hWndClient, WM_DDE_ACK, hWnd, MAKELONG(uAck, aItem)) )
    {                                                              //
      GlobalDeleteAtom( aItem );                              //
    }                                                              //
                                                                   //
    GlobalDeleteAtom( aFileMask );                           //
}
```

Note:

FUNCTION: ListDDERequest(HWND , unsigned , WORD , LONG)

PURPOSE:

Processes and acknowledges a DDE_REQUEST message containing the request for the file list.

```
void FAR ListDDERequest(HWND hWnd, unsigned uMessage, WORD wParam, LONG lParam)
{
    WORD     wFormat    =  LOWORD(lParam);                    //
    WORD     wStatus    =  DDE_NACK;                          //
   ATOM      aItem      =  HIWORD(lParam);                    //
    ATOM     aGetList   = GlobalAddAtom( zGetList );     // Test Atom
                                                               //
    if( wFormat == CF_DSPTEXT && aItem == aGetList )    //
    {                                                          //
      StartFileConversation();                          // Start 2nd conversation
    }                                                          //
    else                                                       // Request Error!
    {                                                          // NACK it and terminate
      PostMessage(wParam,                                      // Callers Handle
                  WM_DDE_ACK,                                  // ACK Message
                  hWnd,                                        // Our Window Handle
                  MAKELONG(wStatus, aItem) );                  // Status / Item
                                                               //
                                                               //  Terminate
      PostMessage(wParam,                                      // Callers Handle
                  WM_DDE_TERMINATE,                            // ACK Message
                  hWnd,                                        // Our Window Handle
                   L );                                        // Status / Item
    }                                                          //
                                                               //
    DestroyWindow( hWndHidden );                               // Terminate LISTFILE
```

```
   DestroyWindow( hGlobalWnd );                                    //
                                                                     //
   GlobalDeleteAtom( aGetList );                                  //
}
```

Note:

FUNCTION: ListDDETerminate(HWND , unsigned , WORD , LONG)

PURPOSE:

Processes a DDE_TERMINATE message from the client and sends the client a DDE_TERMINATE message to complete the DDE termination process.

```
void FAR ListDDETerminate(HWND hWnd, unsigned uMessage, WORD wParam, LONG lParam
)
{
   if( hDDEClient != NULL )                                        //Is client alive?
   {                                                                 //Terminate  him
      PostMessage(hDDEClient, WM_DDE_TERMINATE, hWnd,  L);
      hDDEClient = NULL;                                           //Reset handle
   }
}

//############################################################################

//
//
//   FUNCTION:   StartFileConversation(void)
//
//   PURPOSE:
//
//    This function is called from ListDDERequest where ListFile receives the
//    request for the file list. The secondary DDE conversation is initiated
//    in this function.
//
//

BOOL StartFileConversation(void)
{
   BOOL              bRC = FALSE;                                  //Error until proven otherwise
  ATOM               aIBM327 ,aSESSION,aKEYS;
  LPKEYSTROKES  lpData;

   uCurFileDDEMsg =  ;                                            //current file dde message

                                                                    //
   aIBM327  = GlobalAddAtom( zIBM327  );                          // Create atoms for
   aSESSION = GlobalAddAtom( zSESSION );                           // DDE traffic
   aKEYS       = GlobalAddAtom( zKEYS       );                    // DDE traffic
                                                                    //
   uCurFileDDEMsg = WM_DDE_INITIATE;                                //
                                                                    //
   SendMessage( -1,                                               // Broadcast to all windows
                     uCurFileDDEMsg,                               // initiate session
                     hWndHidden,                                   //  conversation
                     MAKELONG(aIBM327 ,  aSESSION));    //
```

```
                                                        //
  if (hDDEServer == NULL)                               //did we get a DDE link up
  {                                                      //No, better handle error
      bRC = FALSE;                                       //
  }                                                     //
  else                                                  //yes,poke the LISTFILE
  {
      if( (hData = GlobalAlloc(DDE_GLOBAL,(DWORD)sizeof(KEYSTROKES))) != NULL )
      {
          if( (lpData = (LPKEYSTROKES) GlobalLock(hData)) != NULL ) //
          {
              lpData->fRelease        =  TRUE;
              lpData->cfFormat        =  CF_DSPTEXT;
              lpData->uTextType       =  WC_HLLAPITEXT;
               lpData->uRetryCount = 1;

               lstrcpy(lpData->szKeyData,HLLAPI_CLEAR);    //construct listfile string
               lstrcat(lpData->szKeyData,zListFile);          //with hllapi commands
               lstrcat(lpData->szKeyData,Z_SPACE);              //"@C LISTFILE   TEXT A @E"
              lstrcat(lpData->szKeyData,zHostFileMask);  //
              lstrcat(lpData->szKeyData,HLLAPI_ENTER);    //

               GlobalUnlock(hData);                                 // Unlock the buffer

               uCurFileDDEMsg = WM_DDE_POKE;
              PostMessage(hDDEServer,uCurFileDDEMsg,hWndHidden,
                                MAKELONG(hData,aKEYS));
               bRC = TRUE;
          }
          else                                                      //bad  lock
          {
              GlobalFree(hData);
          }
      }
      else                                                      //bad  alloc
      {
      }
  }

  return( bRC );
}
```

Note:

FUNCTION: FileDDEAck(HWND , unsigned , WORD , LONG)

PURPOSE:

Processes DDE_ACK messages from the PC/327 session and, if needed, requests the presentation space from the session by sending a DDE_REQUEST message.

```
void FAR FileDDEAck(HWND hWnd, unsigned uMessage, WORD wParam, LONG lParam)
{
  ATOM      aPS;

   if (uCurFileDDEMsg == WM_DDE_INITIATE)
   {                                                  // Are we trying to Init DDE?
       hDDEServer = wParam;                           // yes, save handle.
      GlobalDeleteAtom(LOWORD(lParam));        //  DELETE  INITIATE  ATOMS
      GlobalDeleteAtom(HIWORD(lParam));        //
       uNumberOfFiles =  ;                           //
   }
   else if(uCurFileDDEMsg == WM_DDE_POKE)      // from listfile cmd or clear
   {
      GlobalDeleteAtom(HIWORD(lParam));        //  DELETE  POKE  ATOM
       if(LOWORD(lParam) < DDE_ACK)                 // NACK
       {
          GlobalFree(hData);                          // free POKE data
          uCurFileDDEMsg = WM_DDE_TERMINATE;    // end conversation
          PostMessage(hDDEServer,uCurFileDDEMsg, hWnd,  L);
          hDDEServer = NULL;
       }
      else                                             //  ACK
       {
          aPS = GlobalAddAtom((LPSTR) zPS);      // presentation space atom
          uCurFileDDEMsg = WM_DDE_REQUEST;       // request ps
          PostMessage(hDDEServer, uCurFileDDEMsg, hWnd, MAKELONG(CF_TEXT, aPS));
       }
   }
   else if(uCurFileDDEMsg == WM_DDE_REQUEST)
   {                                                  // if we get this it is a NACK
       uCurFileDDEMsg = WM_DDE_TERMINATE;       // so end conversation
       PostMessage(hDDEServer,uCurFileDDEMsg, hWnd,  L);
       hDDEServer = NULL;
   }
}
```

Note:

FUNCTION: FileDDEData(HWND , unsigned , WORD , LONG)

PURPOSE:

Processes DDE_DATA messages from the session and parses
the file names from the presentation space buffer returned with the
message.

```
void FAR FileDDEData(HWND hWnd, unsigned uMessage, WORD wParam, LONG lParam)
{
   HANDLE              hPS      = LOWORD(lParam);              // presentation space
   ATOM                aItem = HIWORD(lParam);              // aPS

   ATOM                aKEYS, aPS, aGetList;
   unsigned char zRows[5], zCols[5];
   WORD                uRows, uCols, uRetCode, uBytesInFileList;
  LPPS_CF_TEXT   lpPS;
  LPKEYSTROKES   lpData;
  LPLISTFILES     lpList;

   if( hPS != NULL )
   {
        if((lpPS = (LPPS_CF_TEXT) GlobalLock(hPS)) != NULL)
        {
            lstrncpy(zRows,lpPS->PSROWS,4);
            lstrncpy(zCols,lpPS->PSCOLS,4);
             zRows[4] = '\ ';
             zCols[4] = '\ ';
             uRows = atoi(zRows);
             uCols = atoi(zCols);
             switch( ParsePS(lpPS->PS,uRows,uCols) )
             {
                 case  PSDATA_MORE:
                     if( (hData = GlobalAlloc(DDE_GLOBAL, (DWORD)sizeof(KEYSTROKES))) !=
 NULL )
                     {
                         if( (lpData = (LPKEYSTROKES) GlobalLock(hData)) != NULL )
                         {
                             lpData->fRelease        = TRUE;                        // poke hllapi clear
                             lpData->cfFormat        = CF_DSPTEXT;                  // the ack will post
                             lpData->uTextType       = WC_HLLAPITEXT;               // the next request
                             lpData->uRetryCount = 1;                              //
                            lstrcpy(lpData->szKeyData,HLLAPI_CLEAR);    //

                             GlobalUnlock(hData);                                  // Unlock the buffer

                             aKEYS = GlobalAddAtom((LPSTR) zKEYS       );    // DDE atom
                             uCurFileDDEMsg = WM_DDE_POKE;                         //
                            PostMessage(hDDEServer,uCurFileDDEMsg,hWnd,
                                        MAKELONG(hData,aKEYS));              //
                         }                                                        //
                         else                                                     // bad lock
                         {                                                        //
                             GlobalFree(hData);                                  //
```

```
                    }                                                   //
                  }                                                     //
                   else                                                 // bad alloc
                  {
                  }
                  break;

                case PSDATA_REREAD:                                     // do not clear, reread PS
                   aPS = GlobalAddAtom((LPSTR) zPS);                    // presentation space atom
                   uCurFileDDEMsg = WM_DDE_REQUEST;                     // request ps
                  PostMessage(hDDEServer,uCurFileDDEMsg,  hWnd,
                              MAKELONG(CF_TEXT,aPS));                   //
                  break;                                                //
                                                                        //
                case PSDATA_END:                                        //post DATA to Receive
                   uBytesInFileList = uNumberOfFiles   SIZEOFFILENAME;
                   if( (hData = GlobalAlloc(DDE_GLOBAL, (DWORD)sizeof(LISTFILES)+uByte
sInFileList+2)) != NULL )
                  {
                      if( (lpList = (LPLISTFILES) GlobalLock(hData)) != NULL )
                      {
                          lpList->fRelease             =  TRUE;
                          lpList->cfFormat             =  CF_DSPTEXT;
                          lpList->iItemsInList      =  uNumberOfFiles;
                           lpList->uFilenameWidth = SIZEOFFILENAME;

                           _fmemcpy(lpList->Data, zListFileNames, uBytesInFileList);

                          lpList->Data[uBytesInFileList]      =  NULL_CHAR;
                           lpList->Data[uBytesInFileList+1] = NULL_CHAR;

                          GlobalUnlock(hData);

                           aGetList = GlobalAddAtom((LPSTR) zGetList); // DDE atom

                           if( !PostMessage(hDDEClient,WM_DDE_DATA,hWnd, MAKELONG(hData,
aGetList)) )
                          {
                              GlobalDeleteAtom( aGetList ); // Error, delete atom
                          }

                           //    Do the terminate logic

                           PostMessage(hDDEClient, WM_DDE_TERMINATE, hWnd,  L);
                           hDDEClient = NULL;                           // reset handle
                           PostMessage(hDDEServer, WM_DDE_TERMINATE, hWnd,  L);
                           hDDEServer = NULL;                           // reset handle
                      }                                                 //
                       else                                             // BAD LOCK
                      {                                                 //
                          GlobalFree(hData);                           //
                      }                                                 //
                  }                                                     //
                   else                                                 // BAD ALLOC
                  {                                                     //
                  }                                                     //
                  break;                                                //
                                                                        //
```

```
            default:                                                    //  PSDATA_ERROR
               //error  case                                           //
                //do the terminate logic                              //
               break;                                                   //
         }                                                              //
                                                                        //
         if(lpPS->fAckReq)                                              // ACK required?
         {                                                              //
            PostMessage(hDDEServer, WM_DDE_ACK, hWnd,                   //
                        MAKELONG(DDE_ACK,  aItem));                     //
         }                                                              //
         else                                                           //
         {                                                              //
            GlobalDeleteAtom(aItem);                                    //
         }                                                              //
         if(lpPS->fRelease)                                             // release required?
         {                                                              //
            GlobalUnlock(hPS);                                          //
            GlobalFree(hPS);                                            //
         }                                                              //
         else                                                           //
         {                                                              //
            GlobalUnlock(hPS);                                          //
         }                                                              //
      }                                                                 //
      else                                                              // BAD LOCK
      {                                                                 //
         GlobalFree(hPS);                                               // Free the handle
      }                                                                 //
   }                                                                    //
}
```

Note:

FUNCTION: FileDDETerminate(HWND , unsigned , WORD , LONG)

PURPOSE:

Processes a DDE_TERMINATE message from the session and sends the session
a DDE_TERMINATE message to complete the DDE termination process.

```
void FAR FileDDETerminate(HWND hWnd, unsigned uMessage, WORD wParam, LONG lParam
)
{
   if(hDDEServer != NULL)                                   // Is this an unsolicited
   {                                                         // terminate? yes, reply
      uCurFileDDEMsg = WM_DDE_TERMINATE;                     // with a terminate
      PostMessage(hDDEServer, uCurFileDDEMsg, hWnd,  L);
      hDDEServer = NULL;                                     //
   }                                                         //
}
```

```
//
//
//    FUNCTION: lstrncpy( LPSTR , LPSTR , WORD )
//
//   PURPOSE:
//
//    Long pointer strncpy function.
//
//
//

LPSTR lstrncpy( LPSTR lpzString1, LPSTR lpzString2, WORD uLength )
{
  int  i;
   LPSTR p = lpzString1;
  uchar  ch;

   for( i =  ; i < uLength; i++)
   {
       ch =  (lpzString2++);
        (lpzString1++) = ch;
       if( ch == NULL )
      {
          return( p );                          /  String was less than uLength in Length  /
       }
   }

    (lpzString1) = NULL;                        /  copied up to uLength, add Null on End  /

   return( p );
}

//
//
//    FUNCTION: ParsePS(LPSTR , WORD , WORD )
//
//   PURPOSE:
//
//   COMMENTS:
//
//    Each line of a CF_TEXT PS ends in  x D  x A so the lines are
//    actually uCols+2 characters long.
//
//   RETURNS:
//
//      PSDATA_ERROR
//      PSDATA_END       1
//      PSDATA_MORE      2
//      PSDATA_REREAD  3
//
//
```

```
WORD ParsePS(LPSTR lpPS, WORD uRows, WORD uCols)
{                                                                    //
  WORD uRetCode = PSDATA_MORE;                                          //
  WORD  uOffset   =   ;                                              //
  WORD  uLine       =   ;                                              //
  WORD  uLineStart;                                                  //
  BOOL  bReady      =  FALSE;                                           //
                                                                     //
  if( bFirstScreen )                                          // First Screen
  {                                                                  // Yes, IGNORE first line
     uLineStart = 1;                                              //
     bFirstScreen = FALSE;                                        // reset flag
  }                                                                  //
  else                                                              //  Otherwise
  {                                                                  //
     uLineStart =  ;                                              // Start with first
  }                                                                  //  line
                                                                     //
  for(uLine = uLineStart; uLine < uRows; uLine++)          // check each line of the PS
  {                                                                  // to see if it contains a
     uOffset = uLine   (uCols + 2);                          // filename or Ready
                                                                     //
    lstrncpy(zMsgBuffer,lpPS+uOffset,uCols);                 //
     zMsgBuffer[uCols] = '\ ';                                 // copy & make into str
                                                                     //
     if( ( (lpPS+uOffset+8)    == SPACE_CHAR) &&                 // If there are spaces
           ( (lpPS+uOffset+9)    != SPACE_CHAR) &&                 // at cols 8, 17, & 2
           ( (lpPS+uOffset+17) == SPACE_CHAR) &&                 // non spaces at cols  ,
           ( (lpPS+uOffset+18) != SPACE_CHAR) &&                 // 9, & 18 then the line
           (uNumberOfFiles < MAXFILENAMES) )                        // is a file name.
    {                                                                //
        lstrncpy(&zListFileNames[uNumberOfFiles][ ],        // add found name to
                    lpPS+uOffset,SIZEOFFILENAME);                   // names array
       zListFileNames[uNumberOfFiles][SIZEOFFILENAME-1]  //
          =  NULL_CHAR;                                                //
       uNumberOfFiles++;                                          //
    }                                                                //
                                                                     //
     if(_fstrstr(zMsgBuffer,Z_READY) != NULL)                 // Check if CMS Ready is
     {                                                                // present indicating end
        bReady = TRUE;                                                // of data.
      }                                                              //
  }

  //-------------------------------------------------------
   //check lower right status message and respond accordingly

   lstrncpy(zMsgBuffer,lpPS+((uRows-1)   (uCols + 2)),uCols);
   zMsgBuffer[uCols] = '\ ';                                 // copy line & make a string
                                                                     //

   if(_fstrstr(zMsgBuffer,Z_MORE) != NULL)                  //
   {                                                                 // here we should send a
       uRetCode = PSDATA_MORE;                                       // clear and read the next
   }                                                                 // screen of names
   else if(_fstrstr(zMsgBuffer,Z_RUNNING) != NULL)    //
   {                                                                 //
      if(bReady)                                                  // running + ready tells us
```

```
      {
         uRetCode = PSDATA_END;
      }
      else
      {
         uRetCode = PSDATA_REREAD;
      }
   }
   else
   {
      uRetCode = PSDATA_ERROR;
   }

   return(uRetCode);
}
```

```
      // we are done
      //
   //
   // this case means cms is
      // not done writing file
         // names to the screen,
      // so we should just
      // reread the screen
   //
      //
         // shouldn't happen
      //
      //
// return status
```

LISTFILE.H File

```
//
//
// LISTFILE.H - File containing all defines for application ListFile
//
//
//
//
// Global function defines

int PASCAL          WinMain(HANDLE, HANDLE, LPSTR, int);

BOOL                RegisterApplicationClass( HANDLE );
BOOL                InitInstance( HANDLE, int );

long FAR PASCAL ListFileWndProc(HWND, unsigned, WORD, LONG);
long FAR PASCAL FileListWndProc(HWND, unsigned, WORD, LONG);

BOOL FAR PASCAL About(HWND, unsigned, WORD, LONG);

BOOL FAR            InitDDE( void );

void FAR            ListDDEInitiate(    HWND, unsigned, WORD, LONG );
void FAR            ListDDEPoke(        HWND, unsigned, WORD, LONG );
void FAR            ListDDERequest(     HWND, unsigned, WORD, LONG );
void FAR            ListDDETerminate( HWND, unsigned, WORD, LONG );

void FAR            FileDDEAck(         HWND, unsigned, WORD, LONG );
void FAR            FileDDEData(        HWND, unsigned, WORD, LONG );
void FAR            FileDDETerminate( HWND, unsigned, WORD, LONG );

//
// Dialog Identifiers

#define ID_ABOUT     5
#define ID_SEND      5  1
#define ID_RECV      5  2
#define ID_CONV      5  3
#define ID_IGNORE    5999


//
// Resource Identifiers

#define LISTMENU     9
#define LISTICON     9  1
#define ABOUTBOX     9  2

//
// Load String Identifiers

#define LS_COMMAND_LINE_ERROR1    1
#define LS_TITLE_TEXT             2
#define LS_DDE_TOKENS             3
#define LS_16STR_FORMAT           5
```

```
//
// MISC

#define uchar                unsigned char

#define DDE_GLOBAL    ( GHND | GMEM_DDESHARE ¦ GMEM_MOVEABLE | GMEM_ZEROINIT )

#define DDE_ACK              x8
#define DDE_NACK             x              /  Negative ACK              /
#define DDE_NACK_BUSY      x4              /  Negative ACK/BUSY         /

#define WC_PURETEXT
#define WC_HLLAPITEXT      1

#define SPACE_CHAR          ' '
#define NULL_CHAR           '\ '

#define Z_SPACE             " "
#define HLLAPI_CLEAR        "@C"
#define HLLAPI_ENTER        "@E"
#define Z_MORE              "MORE"
#define Z_RUNNING           "RUNNING"
#define Z_READY             "Ready"

#define PSDATA_ERROR
#define PSDATA_END          1
#define PSDATA_MORE         2
#define PSDATA_REREAD       3

#define SIZEOFFILENAME    21
#define MAXFILENAMES        1

//
// typedef's used by listfile and session

typedef struct tagGetList
            {                                          //Structure used to pass
            unsigned        Unused:13;                 //host file mask to listfile.
            unsigned        fRelease:1;
            unsigned        Unused1:2;
            int             cfFormat;
            int             iSession;

            int             iSystem;
            char            DataSet[41];
            BYTE            Data[1];
            }  GETLIST;

typedef GETLIST FAR      LPGETLIST;

typedef struct tagListFile
            {                                          //Structure used to pass
            unsigned        Unused:13;                 //list of host file names
            unsigned        fRelease:1;                //back to session.
            unsigned        Unused1:2;
            int             cfFormat;
            int             iItemsInList;
            WORD            uFilenameWidth;
```

```
        BYTE            Data[1];
        }  LISTFILES;

typedef LISTFILES FAR  LPLISTFILES;

typedef struct tagKeystrokes
        {                                       //Structure used to poke
        unsigned        unused:12;              //keystrokes to session.
        unsigned        fRelease:1;
        unsigned        freserved:3;
        int             cfFormat;
        unsigned  int   uTextType;
        unsigned  int   uRetryCount;
        unsigned char szKeyData[64];
        } KEYSTROKES, FAR  LPKEYSTROKES;

typedef struct tagPS_CF_TEXT
        {                                       //Structure of a presentation
        unsigned        Unused:12;              //space  in  the  CF_TEXT  format.
        unsigned        fResponse:1;
        unsigned        fRelease:1;
        unsigned        reserved:1;
        unsigned        fAckReq:1;
        int             cfFormat;
        unsigned  char  PSSIZE[4];
        unsigned  char  Tab1[1];
        unsigned  char  PSROWS[4];
        unsigned  char  Tab2[1];
        unsigned  char  PSCOLS[4];
        unsigned  char  Tab3[1];
        unsigned  char  PS[1];
        unsigned  char  Tab4[1];
        unsigned  char  PSFldCount[4];
        unsigned  char  Tab5[1];
        } PS_CF_TEXT, FAR  LPPS_CF_TEXT;
```

LISTDATA.H File

```
//
//
// LISTDATA.H - Data Global to ListFile application
//
//

#ifdef MAIN
#define DECLARE
#define INIT_TO(init_value)    = init_value
#else
#define DECLARE extern
#define INIT_TO(init_value)
#endif

DECLARE HANDLE              hDDEClient                  INIT_TO(NULL);
DECLARE HANDLE              hDDEServer                  INIT_TO(NULL);
DECLARE HANDLE              hGlobalInst;
DECLARE HANDLE              hData;

DECLARE HWND                hGlobalWnd;
DECLARE HWND                hWndHidden;

DECLARE BOOL                bFirstScreen                INIT_TO(TRUE);

DECLARE WORD                uCurFileDDEMsg;
DECLARE WORD                uNumberOfFiles              INIT_TO( );

DECLARE uchar               ListFileClassName       INIT_TO("ListFile");
DECLARE uchar               FileListClassName       INIT_TO("FileList");
DECLARE uchar               zMsgBuffer[257];
DECLARE uchar               zFmtBuffer[129];
DECLARE uchar               zCmdLineSessionID[9];
DECLARE uchar               zHostFileMask[129];

DECLARE uchar               zListFileName
s[MAXFILENAMES+1][SIZEOFFILENAME];
```

LISTFILE.DEF File

```
NAME                LISTFILE

DESCRIPTION         'ListFile Sample Application Copyright (c) 1991, IBM Corp.'

EXETYPE             WINDOWS

STUB                'LISTSTUB.EXE'

CODE                PRELOAD MOVEABLE DISCARDABLE

DATA                PRELOAD MOVEABLE MULTIPLE

HEAPSIZE            4 96
STACKSIZE           512

EXPORTS
  ListFileWndProc  @1
  FileListWndProc  @2
  About            @3
```

LISTFILE.RC File

```
#include "windows.h"
#include "listfile.h"

LISTICON ICON listfile.ico

LISTMENU MENU
BEGIN
  POPUP      "&Help"
  BEGIN
     MENUITEM "&About ListFile ...",        ID_ABOUT
  END
END

STRINGTABLE
BEGIN
   LS_COMMAND_LINE_ERROR1, "ListFile command line must specify a session ID, 'LIS
TFILE /A'"
   LS_TITLE_TEXT,                "ListFile Sample Application Session: "
   LS_DDE_TOKENS,                  "327 LIST LISTFILE FILEMASK GETLIST IBM327  SESSION KE
YS PS"
   LS_16STR_FORMAT,              "%s %s %s %s %s %s %s %s %s %s %s %s %s %s %s %s"
END

ABOUTBOX DIALOG 22, 17, 16 , 1
STYLE DS_MODALFRAME | WS_CAPTION | WS_SYSMENU
CAPTION "About ListFile"
BEGIN
   CONTROL "ListFile DDE Sample Application",              ID_IGNORE,      "static", SS_CE
NTER              | WS_CHILD,      5,    1 ,  15 ,    1
  CONTROL  "&OK",                                                IDOK,             "button",  BS_DE
FPUSHBUTTON     | WS_CHILD,   5 ,   8 ,   6 ,   15
END
```

Appendix E. EHLLAPI Migration

Migrating HLLAPI Applications from DOS to Windows

Some changes must be made to programs written for DOS-mode HLLAPI so that they can run in the Windows environment. In most cases, the changes are code, rather than design changes.

The EHLLAPI application for Windows mode has the following differences from DOS mode:

All Windows EHLLAPI applications must be compiled and linked to run in the Windows protect mode.

Absolute addressing is not allowed. Any application that uses this technique must be modified.

Requests for EHLLAPI services are changed from an interrupt system to a call/return interface.

Windows-mode EHLLAPI includes several new return codes that are not part of DOS-mode EHLLAPI. You can improve error handling in your existing application by adding checks for these new return codes.

Customized LIMs are not supported.

DOS-Mode EHLLAPI Compatibility with Other Emulation Products

EHLLAPI applications that are written for the following IBM HLLAPI products can be used with PC/3270 EHLLAPI.

IBM Workstation Program HLLAPI

IBM Entry Emulator HLLAPI

OS/2 Extended Edition Communications Manager (OS/2 EE) EHLLAPI.

Specific information needed to adapt applications written for each of these IBM HLLAPI products for use with PC/3270 EHLLAPI for DOS mode is contained under a separate heading in this appendix.

IBM Workstation Program HLLAPI

The following differences exist between IBM Workstation Program HLLAPI and PC/3270 EHLLAPI.

Additional Functions

In addition to the functions that are supported by IBM Workstation Program HLLAPI, the following function is supported by PC/3270 EHLLAPI. If desired, this function can be added to your existing application.

Set Cursor (40) function places the cursor at a specified location in the connected host presentation space.

© Copyright IBM Corp. 1996

Functions not Supported

The following functions, supported by IBM Workstation Program HLLAPI, are not supported by PC/3270 EHLLAPI. If your application uses one or more of these functions, changes are required to prevent execution errors. If one of these functions is called from PC/3270 EHLLAPI, return code 10 is generated. Execution is not terminated.

Define Presentation Space (35)

Switch Presentation Space (36)

Display Cursor (37)

Display Presentation Space (38)

Delete Presentation Space (39)

Get 3270 AID Key (54)

Invoke DOS Program (92)

DOS Redirect (93)

WSCTRL (16)

Other Considerations

The function that HLLBASM performs is performed by the customization facility. To allocate memory for your existing application, refer to the *PC/3270 Installation and Administration Guide for Full-Function DOS and Windows Mode*.

PC/3270 EHLLAPI supports additional return codes. If desired, you can add checks for these new return codes to improve error recovery.

Some return codes are deleted or have slightly different meanings. If your application uses these deleted or modified return codes, changes are required to avoid errors or unpredictable results.

It is recommended that you recompile your application and link it with the language interface modules (LIMs) provided with PC/3270 EHLLAPI. If your application uses a customized LIM, the LIM might have to be modified and recompiled. See Appendix B, “EHLLAPI Customized Language Interface Modules.”

PC/3270 EHLLAPI supports only the host presentation space. If your application uses another type, changes are required.

IBM Entry Emulator HLLAPI

The following differences exist between IBM Entry Emulator HLLAPI and PC/3270 EHLLAPI. All the functions that are supported by IBM Entry Emulator HLLAPI are supported by PC/3270 EHLLAPI.

Additional Functions

In addition to the functions that are supported by IBM EHLLAPI, the following functions are supported by PC/3270 EHLLAPI. You can add these functions added to your existing application.

Set Cursor (40) function places the cursor at a specified location in the connected host presentation space.

Start Keystroke Intercept (50) function initiates keystroke monitoring.

Get Key (51) function receives a keystroke.

Post Intercept Status (52) function processes (accepts or rejects) a given keystroke.

Stop Keystroke Intercept (53) function terminates keystroke monitoring.

Other Considerations

The presentation space short name for Entry Emulator HLLAPI was E. The default presentation space short name for PC/3270 EHLLAPI is A. To use programs written for Entry Emulator HLLAPI, change the source code references to PSID.

PC/3270 EHLLAPI supports up to eight sessions. If desired, you can add calls to more than one session to your application.

PC/3270 EHLLAPI supports additional return codes. If desired, you can add checks for these new return codes to improve error recovery.

Some return codes are deleted or have slightly different meanings. If your application uses these deleted or modified return codes, changes are required to avoid errors or unpredictable results.

It is recommended that you recompile your application and link it with the language interface modules (LIMs) provided with PC/3270 EHLLAPI. If your application uses a customized LIM, the LIM might have to be modified and recompiled. See Appendix B, “EHLLAPI Customized Language Interface Modules.”

IBM OS/2 Extended Edition HLLAPI

The following differences exist between IBM OS/2 Extended Edition HLLAPI and PC/3270 EHLLAPI. All the functions that are supported by IBM OS/2 Extended Edition HLLAPI are supported by PC/3270 EHLLAPI.

Considerations

The presentation space short name for OS/2 Extended Edition HLLAPI is A through H, and the default is A. The default presentation space short name for PC/3270 EHLLAPI is A.

If your OS/2 Extended Edition application used the direct call format, convert your calls to indirect call format and link with a suitable LIM.

The trace facility for PC/3270 EHLLAPI is different from the trace facility for OS/2 Extended Edition HLLAPI. PC/3270 EHLLAPI messages are sent to the screen, and not to a file. The messages that can be returned by the PC/3270 EHLLAPI trace facility are described in Chapter 3, “EHLLAPI Programming Overview” and Appendix A, “EHLLAPI Messages.”

Some return codes are deleted or have slightly different meanings. If your application uses these deleted or modified return codes, changes are required to avoid errors or unpredictable results.

It is required that you recompile your application in the DOS mode and link it with the language interface modules (LIMs) provided with PC/3270 EHLLAPI. If your application uses a customized LIM, the LIM might have to be modified and recompiled. See Appendix B, "EHLLAPI Customized Language Interface Modules."

If your application calls OS/2 functions, these calls must be converted to DOS calls. See the related operating system manuals.

Differences in memory allocation and file handling techniques between DOS and OS/2 might affect the Send File (90) and Receive File (91) functions. If your application uses these functions, refer to "SEND.EXE and RECEIVE.EXE Files" on page 32 and "Considerations for Using Send File (90) and Receive File (91) Functions (PC/3270 DOS Mode Only)" on page 31 to see if modification is required.

Appendix F. Query Reply Data Structures Supported by EHLLAPI

This appendix lists and defines the query reply structures supported by the EHLLAPI structured field interface for PC/3270. See the *IBM 3270 Information Display System Data Stream Programmer's Reference* or, in the case of an IBM licensed program, the documentation for the specific licensed program.

Notes:

1. EHLLAPI must scan the query reply buffers to locate the destination or origin ID (DOID) self-defining parameter (SDP) for the structured field support to work reliably. The DOID field is then filled in with the assigned ID.
2. The application should build the query reply data structures in the application's private memory.
3. Only cursory checking is performed on the query reply data. Only the ID and the length of the structure are checked for validity.
4. The 2-byte length field at the beginning of each query reply **is not byte-reversed**.
5. Only one distributed data management (DDM) base-type connection is allowed per host session. If the DDM connection supports the SDP for the DOID, then multiple connections are allowed.
6. If a nonzero return code is received indicating that an application is already connected to the selected session (RC 32 or 39), use that presentation space with caution. Conflicts with SRPI, file transfer, and other EHLLAPI applications might result.

The DDM Query Reply

Several DDM query reply formats are supported. Some of these formats are as follows:

Table 22. DDM Query Reply Base Format

Offset	Length	Content	Meaning
0	1 word	Length	Length of structure
2	1 byte	X'81'	Query reply ID
3	1 byte	X'95'	Query reply type
4–5	2 bytes	FLAGS	Reserved
6–7	2 bytes	LIMIN	Maximum DDM bytes allowed in inbound transmission
8–9	2 bytes	LIMOUT	Maximum DDM bytes allowed in outbound transmission
10	1 byte	NSS	Number of subsets identifier
11	1 byte	DDMSS	DDM subset identifier

© Copyright IBM Corp. 1996

DDM Application Name Self-Defining Parameter

The DDM Application Name self-defining parameter provides the host application with the name of the application containing control of the DDM Auxiliary Device. The controlling application is identified by the DOID in the Direct Access self-defining parameter.

This SDP is optional, but it is necessary if a host application is to identify a distinct DDM auxiliary device when more than one application is in existence at a remote workstation.

Table 23. DDM Application Name SDP

Offset	Length	Content	Meaning
0	1 byte	Length	Parameter length
1	1 byte	X'02'	DDM application name
2–n	n–2 bytes	NAME	Name of the remote application program

NAME. The name has eight characters or less and is the means by which a host application can relate to an application in a remote workstation. It is the responsibility of the host and remote application users to ensure that the name is understood by the application at each end.

PCLK Protocol Controls Self-Defining Parameter

The PCLK protocol control self-defining parameter indicates that the PCLK protocol control structured field, ID = X'1013', can be used for both inbound and outbound data streams destined to or from the DDM auxiliary device processor.

Table 24. DDM PCLK Auxiliary Device SDP

Offset	Length	Content	Meaning
0	1 byte	X'04'	Parameter length
1	1 byte	X'03'	PCLK protocol controls
2–3	2 bytes	VERS	Protocol version

VERS. The value given in VERS is used to indicate the versions of PCLK installed in the terminal at the time the query reply is returned. For example, X'0001' indicates PCLK version 1.1.

See the *IBM 3270 Information Display System Data Stream Programmer's Reference* for the field definitions for this query reply.

Base DDM Query Reply Formats

The following query reply formats are *examples* of some of the Base + SDP combinations possible. Not all of the combinations are shown.

Table 25. Base DDM Query Reply Format with Name and Direct Access SDPs

Offset	Length	Content	Meaning
0	1 word	Length	Length of structure (includes SDPs)
2	1 byte	X'81'	Query reply ID
3	1 byte	X'95'	Query Reply type
4–5	2 bytes	FLAGS	Reserved
6–7	2 bytes	LIMIN	Maximum DDM bytes allowed in inbound transmission
8– 9	2 bytes	LIMOUT	Maximum DDM bytes allowed in outbound transmission
10	1 byte	NSS	Number of subsets supported
11	1 byte	DDMSS	DDM subset identifier
12	1 byte	Length (n+2)	Parameter length
13	1 byte	X'02'	DDM Application name
14–(13+n)	n bytes	Name	Name of the remote application program
14+n	1 byte	X'04'	Parameter length
15+n	1 byte	X'01'	Direct Access ID
16+n–17+n	2 bytes	DOID	Destination or origin ID assigned by the subsystem

The SDPs begin at offsets 12 and (14+n) where n is the length of the application name supplied at offset 14.

See the *IBM 3270 Information Display System Data Stream Programmer's Reference* for the field definitions for this query reply.

Table 26. Base DDM Query Reply Format with Direct Access and Name SDPs

Offset	Length	Content	Meaning
0	1 word	Length	Length of structure (includes SDPs)
2	1 byte	X'81'	Query reply ID
3	1 byte	X'95'	Query reply type
4–5	2 bytes	FLAGS	Reserved
6–7	2 bytes	LIMIN	Maximum DDM bytes allowed in inbound transmission
8–9	2 bytes	LIMOUT	Maximum DDM bytes allowed in outbound transmission
10	1 byte	NSS	Number of subsets supported
11	1 byte	DDMSS	DDM subset identifier
12	1 byte	X'04'	Parameter length
13	1 byte	X'01'	Direct access ID
14–15	2 bytes	DOID	Destination or origin ID assigned by the subsystem
16	1 byte	Length (n+2)	Parameter length
17	1 byte	X'02'	DDM application name
16+n–(17+n)	n bytes	Name	Name of the remote application program

The SDPs begin at offsets 12 and 16.

See the *IBM 3270 Information Display System Data Stream Programmer's Reference* for the field definitions for this query reply.

The IBM Auxiliary Device Query Reply

The auxiliary device reply is used to indicate to the host application the support of an IBM auxiliary device that uses a data stream defined by IBM, see the *IBM 3270 Information Display System Data Stream Programmer's Reference* for more detail.

When the function is supported, the query reply is transmitted inbound in reply to a Read Partition structured field specifying Query or Query List (QCODE List = X'9E', Equivalent, or All).

When a workstation supports multiple auxiliary devices, the IBM Aux Device query reply must be sent for each device.

Optional Parameters:

All parameters shown in the “base” part of the query reply must be present. Parameters not used are set to X'00'.

At least one self-defining parameter must be present.

Table 27. IBM Auxiliary Device Base Format with Direct Access SDP

Offset	Length	Content	Meaning
0–1	1 word	Length	Length of structure (includes SDPs)
2	1 byte	X'81'	Query reply ID
3	1 byte	X'9E'	IBM Aux Dev reply
4	1 byte	FLAGS	Reserved
	BIT 0	QUERY B'1'	Read Part (Query, Query List) Aux. Dev. Supports Query
	1–7	RES	Reserved, must be B'0's
5	1 byte	FLAGS	Reserved
6–7	2 bytes	LIMIN	Maximum DDM bytes allowed in inbound transmission
8–9	2 bytes	LIMOUT	Maximum DDM bytes allowed in outbound transmission
10	1 byte	TYPE X'01' X'02' Others	Type of Auxiliary device supported IBM Auxiliary Device Display IBM Auxiliary Device Printer Reserved
11	1 byte	X'04'	Parameter length
12	1 byte	X'01'	Direct Access
13–14	1 word	DOID	Destination or origin ID assigned by the subsystem

QUERY

This bit must be set to B'1' for all IBM auxiliary devices to indicate that it supports receiving a Read Partition (Query, Query List). The host applications can then use a Read Partition directed to the auxiliary device to determine its characteristics. The destination or origin structured field is used to direct the Read Partition structured field to the auxiliary device.

The minimum support level for the IBM auxiliary device is to return the Null query reply in response to the Read Partition.

LIMIN

States the maximum number of bytes that can be sent in an inbound transmission. A LIMIN value of X'0000' indicates no implementation limit on the number of bytes transmitted inbound.

LIMOUT

States the maximum number of bytes that can be sent to IBM auxiliary device in an outbound transmission. A LIMOUT value of X'0000' indicates no implementation limit on the number of bytes transmitted outbound.

TYPE

Identifies the auxiliary device being supported. Two values are valid. One identifies an auxiliary display, and the other identifies an auxiliary printer. All other values are reserved.

The IBM Auxiliary Device processor supports two Self-Defining Parameters, 01 and 03. These are defined in Table 28 on page 396.

Direct Access Self-Defining Parameter

This self-defining parameter provides the ID for use in the destination/origin structured field in the direct access of the IBM auxiliary device.

This SDP is always required to accompany the base query reply.

Table 28. IBM Auxiliary Device Direct Access SDP

Offset	Length	Content	Meaning
0	1 byte	X'04'	Parameter length
1	1 byte	X'01'	Direct access ID
2–3	2 bytes	DOID	Destination or origin ID

DOID. The value in these bytes is used in the ID field of the destination/origin structured field to identify the auxiliary device as the destination or origin of the data that follows.

PCLK Protocol Controls Self-Defining Parameter

The presence of the PCLK Protocol Control self-defining parameter indicates that the PCLK Protocol Control structured field, ID = X'1013', can be used for both inbound and outbound in data streams destined to or from the IBM Auxiliary Device processor.

Table 29. IBM Auxiliary Device PCLK SDP

Offset	Length	Content	Meaning
0	1 byte	X'04'	Parameter length
1	1 byte	X'03'	PCLK protocol controls
2–3	2 bytes	VERS	Protocol version

VERS. The value given in VERS is used to indicate the versions of PCLK installed in the terminal at the time the query reply is returned. For example, X'0001' indicates PCLK version 1.1.

See the *IBM 3270 Information Display System Data Stream Programmer's Reference* for the field definitions for this query reply.

The OEM Auxiliary Device Query Reply

The OEM Auxiliary Device query reply format is as follows:

Table 30. OEM Auxiliary Device Base Format with Direct Access SDP

Offset	Length	Content	Meaning
0–1	1 word	Length	Length of structure (includes SDPs)
2	1 byte	X'81'	Query reply ID
3	1 byte	X'8F'	OEM query reply
4–5	2 bytes	FLAGS	Reserved
6–13	4 words	DTYPE	Device type
14–21	4 words	UNAME	User assigned name
22	1 byte	X'04'	Parameter length
23	1 byte	X'01'	Direct access
24–25	1 word	DOID	Destination or origin ID assigned by the subsystem

See the *IBM 3270 Information Display System Data Stream Programmer's Reference* for the field definitions for this query reply.

The OEM Auxiliary Device processor supports two Self-Defining Parameters, 01 and 03. These are defined in Table 31.

Direct Access Self-Defining Parameter

This self-defining parameter provides the ID for use in the destination/origin structured field in the direct access of the OEM auxiliary device.

Table 31. OEM Auxiliary Device Direct Access SDP

Offset	Length	Content	Meaning
0	1 byte	X'04'	Parameter length
1	1 byte	X'01'	Direct access ID
2–3	2 bytes	DOID	Destination or origin ID

DOID. The value in these bytes is used in the ID field of the destination/origin structured field to identify the auxiliary device as the destination or origin of the data that follows.

PCLK Protocol Controls Self-Defining Parameter

The presence of the PCLK Protocol Control self-defining parameter indicates that the PCLK Protocol Control structured field, ID = X'1013', can be used for both inbound and outbound in data streams destined to or from the OEM Auxiliary Device processor.

Table 32. IBM Auxiliary Device PCLK SDP

Offset	Length	Content	Meaning
0	1 byte	X'04'	Parameter length
1	1 byte	X'03'	PCLK protocol controls
2–3	2 bytes	VERS	Protocol version

VERS. The value given in VERS is used to indicate the versions of PCLK installed in the terminal at the time the query reply is returned. For example, X'0001' indicates PCLK version 1.1.

The Cooperative Processing Requester Query Reply

The cooperative processing requester query reply is also called the SRPI query reply or CPSI query reply. The format is as follows:

Table 33. CPR Query Reply Buffer Format

Offset	Length	Content	Meaning
0	1 word	Length	Length of structure (includes SDPs)
2	1 byte	X'81'	Query reply ID
3	1 byte	X'AB'	Query Reply type
4–5	2 bytes	FLAGS	Reserved
6–7	2 bytes	LIMIN	Maximum DDM bytes allowed in inbound transmission
8–9	2 bytes	LIMOUT	Maximum DDM bytes allowed in outbound transmission
10	1 byte	FETAL	Length (in bytes) of the following feature information
11–12	1 word	FEATS	CPR length and feature flags
13 to (N*2)+12	0–2 bytes	FEATSs	Additional flags
(N*2)+12	1 byte	X'04'	Length of DOID SDP
(N*2)+13	1 byte	X'01'	Type of D/O ID
(N*2)+14	1 word	DOID	Destination or origin ID assigned by the subsystem

See the *IBM 3270 Information Display System Data Stream Programmer's Reference* for the field definitions for this query reply.

The Product-Defined Query Reply

This query reply is used by IBM products using registered subidentifiers within the X'9C' data structure. The Product Defined Data Stream query reply indicates support of a 3270DS workstation auxiliary device that uses an IBM product defined data stream. The data stream is *not* defined by a format architecture document having an identifiable control point such as an architecture review board.

When an auxiliary device supports an IBM product defined data stream, this query reply is transmitted inbound in reply to a Query List (QCODE List = X'9C' or All).

Optional Parameters

All parameters shown in the base part of the query reply and the Direct Access self-defining parameter must be present.

The format of the Product Defined query reply is as follows:

Table 34. IBM Product Defined Query Reply Base Format

Offset	Length	Content	Meaning
0–1	1 word	Length	Length of structure (includes SDPs)
2	1 byte	X'81'	Query reply ID
3	1 byte	X'9C'	IBM product-defined data stream
4–5	2 bytes	FLAGS	Reserved
6	1 byte	REFID	Reference identifier
7	1 byte	SSID	Subset identifier
8	1 byte	X'04'	Parameter length
9	1 byte	X'01'	Direct Access
10–11	1 word	DOID	Destination/origin ID assigned by the subsystem

Valid values for REFID (offset 6) and SSID (offset 7) of the Product Defined query reply are as follows:

Table 35. Valid REFID and SSID Values for the IBM Product Defined Query Reply

REFID	SSID	Product and Data Stream Documentation
X'01'		5080 Graphics System: This reference ID indicates the 5080 Graphics System data stream is supported by the auxiliary device. Descriptions of the 5080 Graphics Architecture, structured field, subset ID, DOID and associated function sets, are defined in the *IBM 5080 Graphics System Principles of Operation*
	X'01' X'02'	5080 HGFD Graphics Subset 5080 RS232 Ports Subset
X'02'		WHIP API (replaced by SRL name when written) This reference ID indicates that the WHIP API data stream is supported by the auxiliary device. A description of the WHIP API architecture is defined in the *IBM RT PC Workstation Host Interface Program Version 1.1 User's Guide and Reference Manual*
	X'01'	WHIP Subset 1
X'03' to X'FF'		All other values are reserved.

The IBM product-defined processor supports only the Direct Access Self-Defining Parameter. It is defined in Table 36 on page 400.

Direct Access Self-Defining Parameter

The presence of the Direct Access ID self-defining parameter indicates the auxiliary device can be accessed directly by using the destination or origin structured field. When multiple auxiliary devices that use a product-defined data stream are used, separate product-defined data stream query replies must be provided, each of which must have a unique DOID.

Table 36. IBM Product-Defined Direct Access SDP

Offset	Length	Content	Meaning
0	1 byte	X'04'	Parameter length
1	1 byte	X'01'	Direct access ID
2–3	2 bytes	DOID	Destination or origin ID

DOID. The value in these bytes is used in the ID field of the destination/origin structured field to identify the auxiliary device as the destination or origin of the data that follows.

The Document Interchange Architecture Query Reply

This query reply indicates the Document Interchange Architecture (DIA) function set supported. The format of the DIA Query Reply is as follows:

Table 37. IBM DIA Base Format

Offset	Length	Content	Meaning
0	1 word	Length	Length of structure (includes SDPs)
2	1 byte	X'81'	Query reply ID
3	1 byte	X'97'	IBM DIA
4–5	2 bytes	FLAGS	Reserved
6–7	2 bytes	LIMIN	Maximum DDM bytes allowed in inbound transmission
8–9	2 bytes	LIMOUT	Maximum DDM bytes allowed in outbound transmission
10	1 byte	NFS	Number of 3-byte function set IDs that follow
11–13	3 bytes	DIAFS	DIA function set identifier
14–(13+(N*3))	N*3 bytes	DIAFSs	Additional DIA function set set IDs
14+(N*3)	1 byte	X'04'	Parameter length
15+(N*3)	1 byte	X'01'	Direct Access
16+(N*3)	1 word	DOID	Destination/origin ID assigned by the subsystem

The DIA Auxiliary Device processor supports only the Direct Access Self-Defining Parameter. It is defined in Table 38 on page 401.

Direct Access Self-Defining Parameter

The presence of the Direct Access ID self-defining parameter indicates the auxiliary device can be accessed directly by using the destination/origin structured field.

Table 38. IBM Product Defined Direct Access SDP

Offset	Length	Content	Meaning
0	1 byte	X'04'	Parameter Length
1	1 byte	X'01'	Direct Access ID
2–3	2 bytes	DOID	Destination/origin ID

DOID. The value in these bytes is used in the ID field of the destination/origin structured field to identify the auxiliary device as the destination or origin of the data that follows.

See the *IBM 3270 Information Display System Data Stream Programmer's Reference* for the field definitions for this query reply.

Index

Numerics

01, Connect Presentation Space 63
02, Disconnect Presentation Space 94
03, Send Key 104, 143, 161, 173
04, Wait 173
05, Copy Presentation Space 81
06, Search Presentation Space 138, 173
07, Query Cursor Location 114
08, Copy Presentation Space to String 85
09, Set Session Parameters 151
10, Query Sessions 120
101, Connect Window Services 65
102, Disconnect Window Service 95
103, Query Window Coordinates 124
104, Window Status 174
105, Change Switch List LT Name 59
106, Change PS Window Name 57
11, Reserve 134
110, Start Playing Macro 164
12, Release 133, 134
120, Connect for Structured Fields 61
121, Disconnect from Structured Fields 93
122, Query Communications Buffer Size 112
123, Allocate Communications Buffer 55
124, Free Communications Buffer 100
125, Get Request Completion 105
126, Read Structured Fields 126
127, Write Structured Fields 177
13, Copy OIA 73
14, Query Field Attribute 115
15, Copy String to Presentation Space 91
17, Storage Manager 31, 168
18, Pause 108, 166
20, Query System 122
21, Reset System 134, 135, 151
22, Query Session Status 68, 118
23, Start Host Notification 31, 108, 117, 159, 169, 170
24, Query Host Update 108, 109, 117, 166
25, Stop Host Notification 166
30, Search Field 136, 173
31, Find Field Position 70, 98
32, Find Field Length 70, 96
33, Copy String to Field 89
34, Copy Field to String 69
40, Set Cursor 150
41, Start Close Intercept 157
42, Query Close Intercept 111
43, Stop Close Intercept 165
50, Start Keystroke Intercept 161
51, Get Key 101, 110, 161
52, Post Intercept Status 110, 161
53, Stop Keystroke Intercept 167
90, Send File 140
91, Receive File 130
99, Convert Position or Convert RowCol 66
99, Convert Position or RowCol
  return code 21

A

AID key mnemonics 34
Allocate Communications Buffer (123) 55
ASCII characters 33
ASCII mnemonics 33
ATTRB 153
attribute byte
  values 21
attribute bytes 69, 81, 85, 89, 92, 153
AUTORESET 154

B

BASIC
  compatible xx
  compiling 38
  Copy Presentation Space (5) considerations 40
  function call syntax 38
  interpretive initialization 39, 40
  linking 38
  memory preallocation for returned data strings 17, 334
  sample error-handling code 334
  sample program 327
BASIC language
  limitations because of
    solutions to 168
beep 110
BLANK 155

C

C
  compatible xx
  compiling 41, 49
  function call syntax 41, 48
  linking 41, 49
  memory preallocation for returned data strings 18, 336
  parameter name declaration 41, 48
  sample error-handling code 341
  sample program 335
call (input) parameters
  general 51

© Copyright IBM Corp. 1996

call (input) parameters *(continued)*
passing 18
call statement
See Function calls
calls, prerequisite 51
Change Menu Item 267
Change PS Window Name (106) 57
Change Switch List LT Name (105) 59
character, escape 34, 102, 144, 154
characters, ASCII 33
COBOL
compatible xx
compiling 42
function call syntax 42
linking 42
memory preallocation for returned data strings 18, 342
parameter name declaration 42
sample error-handling code 348
sample program 341
Code Conversion 192
code, function xi
command highlighting xviii
common error-handling
general 21
return code summary 30
communication services functions
Receive File (91) 130
Send File (90) 140
compatible
DOS versions xx
hardware xx
high-level languages xx
host xxi
related publications xxi
software xx
compile option highlighting xix
compiling
BASIC 38
C 41, 49
COBOL 42
general 22
Pascal 46
composite screen 2
CONLOG 154
Connect for Structured Fields (120) 61
Connect Presentation Space (1) 63, 64, 143
functions where not required 64
interaction with disconnect 26
used in sample program
BASIC 328
C 336
COBOL 345
Pascal 354
Connect Window Services (101) 65
CONPHYS 154
Convert Position or RowCol (99) 66
return code 21
used in sample program
BASIC 330
Copy Field to String (34) 11, 69
copy functions
Copy Field to String (34) 69
Copy OIA (13) 73
Copy Presentation Space (5) 81
Copy Presentation Space to String (8) 85
Copy String to Field (33) 89
Copy String to Presentation Space (15) 91
Copy OIA (13) 8, 73
Copy Presentation Space (5) 81
use with BASIC 40
used in sample program
COBOL 345
Copy Presentation Space to String (8) 8, 85
used in sample program
Pascal 353
Copy String to Field (33) 11, 89
used in sample program
BASIC 332
Copy String to Presentation Space (15) 91
used in sample program
C 340
COBOL 345
create menu item 273
current directory 32
current subdirectory 32
cursor movement 9
customization 25

D

data string
specification 9
data string (calling)
general 18
data string (returned)
general 21
DDE data items
LU Topic 189
Session Topic 189
System Topic 188
DDE functions 187
Code Conversion 192
DDE data items 187
LU Topic 189
Session Topic 189
System Topic 188
Find Field 193
function list 190
Get Keystrokes 195
Get Mouse Input 197

DDE functions *(continued)*
Get Number of Close Requests 200
Get Operator Information Area 201
Get Partial Presentation Space 202
Get Presentation Space 205
Get Session Status 208
Get System Configuration 210
Get System Formats 212
Get System Status 213
Get System SysItems 214
Get System Topics 215
Get Trim Rectangle 216
Initiate Session Conversation 217
Initiate Structured Field Conversation 218
Initiate System Conversation 219
naming conventions for parameters 191
Put Data to Presentation Space 220
Search for String 221
Send Keystrokes 222
Session Execute Macro 223
Set Cursor Position 230
Set Mouse Intercept Condition 231
Set Presentation Space Service Condition 234
Set Session Advise Condition 236
Set Structured Field Service Condition 238
Start Close Intercept 239
Start Keystroke Intercept 241
Start Mouse Input Intercept 242
Start Read SF 246
Start Session Advise 249
Stop Close Intercept 251
Stop Keystroke Intercept 252
Stop Mouse Input Intercept 253
Stop Read SF 254
Stop Session Advise 255
summary 260
Terminate Session Conversation 256
Terminate Structured Field Conversation 257
Terminate System Conversation 258
Write SF 259
DDE menu functions 264
Change Menu Item 267
create menu item 273
Initiate Menu Conversation 275
list 266
Start Menu Advise 276
Stop Menu Advise 278
Terminate Menu Conversation 279
debugging 23
default directory
Receive File 132
Send File 142
device services functions
Get Key (51) 101
Post Intercept Status (52) 110
Release (12) 133
device services functions *(continued)*
Reserve (11) 134
Start Keystroke Intercept (50) 161
Stop Keystroke Intercept (53) 167
directory, current 32
directory, default
Receive File 132
Send File 142
Disconnect from Structured Fields (121) 93
Disconnect Presentation Space (2) 94
interaction with connect 26
used in sample program
BASIC 332
C 339
COBOL 348
Pascal 354
Disconnect Window Service (102) 95
diskette-swapping 32
DLL
introduction 5
DOS SETBLOCK command 31

E

EAB 155
EHLLAPI
interface description 3
running your program 22
EHLLAPI function names xviii
EHLLAPI function numbers xi
EHLLAPI Functions
general 51
EHLLAPI/DDE 1
advantages of using 1
defined 1
overview 1, 5
parts of 5
Programs 327
EHLLAPI/DDE.DLL 5
emulator, terminal 3
EOT 10, 152
error code, extended 122
error-handling
general 30
source code 21
ESC 154
escape character 34, 102, 144, 154
exiting applications 22
extended error code 122

F

field-formatted PS 70, 136
field-related functions
Copy Field to String (34) 69
Copy String to Field (33) 89

field-related functions *(continued)*
Find Field Length (32) 96
Find Field Position (31) 98
Search Field (30) 136
fields, host
input protected 143
numeric only 143
file transfer functions
Receive File (91) 130
Send File (90) 140
Find Field 193
Find Field Length (32) 70, 96
used in sample program
BASIC 332
Find Field Position (31) 70, 98
used in sample program
BASIC 331
FPAUSE 153
Free All Storage subfunction 172
Free Communications Buffer (124) 100
Free Storage subfunction 170
function calls 5
call (input) parameters 51
defined 4
notes on using the function 51
page layout conventions 51
prerequisite calls 51
return (output) parameters 51
source code pattern 19
use of 51
Function code
calling, general 18
introduction 4
numbers xi
returned, general 21
values xi
function code parameter xviii
function names xviii
function numbers xi
functions, by number xi

G

Get Key (51) 33, 101, 110, 161
used in sample program
COBOL 347
Get Keystrokes 195
Get Mouse Input 197
Get Number of Close Requests 200
Get Operator Information Area 201
Get Partial Presentation Space 202
Get Presentation Space 205
Get Request Completion (125) 105
Get Session Status 208
Get Storage subfunction 169
Get System Configuration 210
Get System Formats 212
Get System Status 213
Get System SysItems 214
Get System Topics 215
Get Trim Rectangle 216
graphics, host xx

H

hardware requirements xx
highlighting xix
ASCII mnemonics xix
commands
compile xix
link xix
displayed text xix
entry point names xix
function names xix
keywords xix
operator input xix
options
compile xix
link xix
session xix
programmer input xix
sample code xix
variable names xix
hllapi 19
hllapiw 20
host fields
input protected 143
numeric only 143
host graphics xx
host requirements xxi
host session 25
HWND 20

I

IBM BASIC
See BASIC
IBM C
See C
IBM COBOL
See COBOL
IBM Macro Assembler
See Macro Assembler
IBM Pascal
See Pascal
IBM Support Center 122
initialization
general 17
Initiate Menu Conversation 275
Initiate Session Conversation 217

Initiate Structured Field Conversation 218
Initiate System Conversation 219
input protected fields 143
installation 22, 25
interface description, general 3
IPAUSE 153

K

keyboard mnemonics
  See also “Mnemonics.” 32
keyboard, session 32
keystroke interception, Get Key (51) 101
keyword highlighting xviii

L

length (calling)
  general 18
length (returned)
  general 21
LIM
  introduction 5
LIMs
  Customized 15
  entry point names 16
  Entrypoint name in function calls 19
  filenames 16
  guidelines 15—16
  Provided 15
  user-generated 15, 323
link option highlighting xix
linking
  BASIC 38
  C 41, 49
  COBOL 42
  general 22
  Pascal 46
LWAIT 154, 173

M

Macro Assembler
  assembling 45, 50
  compatible xx
  function call syntax 45, 50
  linking 45
  memory preallocation for returned data strings 18
  parameter control block 43
  passing parameter values in 43
  PCB 43
  PCB rules 43
  pre-call register setting 43
  pre-call syntax, function 17 44
Macro Assembler (Microsoft)
  linking 50
Macro Assembler (Microsoft) *(continued)*
  pre-call register setting 50
memory preallocation 17, 21, 22
memory requirements xx
mnemonics 92
  for Send Key 32
mnemonics, AID key 34
mnemonics, ASCII 33
mnemonics, shift key 33

N

names, EHLLAPI functions xviii
NOATTRB 153
NOBLANK 155
NOEAB 155
NOQUIET 153
NORESET 154
NORESUME 155
notes on using this function
  general 51
NOXLATE 155
numbers, functions xi
numeric only fields 143
NWAIT 154, 173

O

OIA 73, 173
Operator Information Area
  See “OIA.” 73
operator services functions
  Pause (18) 108
  Query Host Update (24) 117
  Query Session Status (22) 118
  Query Sessions (10) 120
  Query System (20) 122
  Reset System (21) 135
  Send Key (3) 143
  Set Session Parameters (9) 151
  Start Host Notification (23) 159
  Stop Host Notification (25) 166
  Wait (4) 173
options 152—156
options, session, highlighting xix

P

parameters
  assigning values 20
  call 51
  data string, call 18
  data string, return 21
  function code xviii
  function code, call 18
  function code, return 21

parameters *(continued)*
general 3
in function call statement 19
length, call 19
length, return 21
passing values 20
PS position, call 19
receiving values 21
return code, return 21
variable name rules 20
Pascal
compatible xx
compiling 46
function call syntax 46
linking 46
memory preallocation for returned data strings 18, 351
parameter name declaration 45
sample error-handling code 352
sample program 349
Pascal string handling 46
path, default
Receive File 132
Send File 142
Pause (18) 8, 108, 166
used in sample program
BASIC 328
C 337
COBOL 346
PCSAPI 181
how to use 181
pcsQueryEmulatorStatus 182
pcsQueryWorkstationProfile 183
pcsStartSession 184
pcsStopSession 185
pcsQueryEmulatorStatus 182
pcsQueryWorkstationProfile 183
PCSSAMP.EXE
See "Program Sampler" 52
pcsStartSession 184
pcsStopSession 185
pogram operator 2
defined 2
tasks 2
Post Intercept Status (52) 13, 110, 161
used in sample program
COBOL 347
preallocation of memory 17, 21, 22
prerequisite calls
general 51
summary 28
presentation services functions
Change PS Window Name (106) 57
Change Switch List LT Name (105) 59
Connect Presentation Space (1) 63
Connect Window Services (101) 65
presentation services functions *(continued)*
Copy Field to String (34) 69
Copy OIA (13) 73
Copy Presentation Space (5) 81
Copy Presentation Space to String (8) 85
Copy String to Field (33) 89
Copy String to Presentation Space (15) 91
Disconnect Presentation Space (2) 94
Find Field Length (32) 96
Find Field Position (31) 98
Get Request Completion (125) 105
Query Cursor Location (7) 114
Query Field Attribute (14) 115
Search Field (30) 136
Search Presentation Space (6) 138
Set Cursor (40) 150
presentation space
character table 74
cursor movement 9
determining size of existing 25
field-formatted 69, 70, 89, 96, 98, 136
host-connected 26
maximum number of 26
how specified 26
identifier 26, 27
blank specifier 27
function 26
how processed 26
letter specifier 27
null specifier 27
processing for functions not requiring connect 27
processing for functions requiring connect 27
size 19, 21, 25
supported types 25
presentation space position (calling)
general 19
program operator 7
Program Sampler 52
installing and running 52
prompt, diskette-swapping 32
PSID
See presentation space, identifier
purpose
EHLLAPI 1
Put Data to Presentation Space 220

Q

Query Close Intercept (42) 111
Query Communications Buffer Size (122) 112
Query Cursor Location (7) 114
used in sample program
BASIC 331
Query Field Attribute (14) 115
Query Free Storage subfunction 171

Query Host Update (24) 108, 109, 117, 159, 166
used in sample program
BASIC 329
C 337
COBOL 346
Query Reply Data Structures
Query Session Status (22) 68, 118
Query Sessions (10) 120
Query System (20) 122
used in sample program
BASIC 334
C 341
COBOL 348
Pascal 352
Query Window Coordinates (103) 124
QUIET 153

R

Read Structured Fields (126) 126
Receive File (91) 11, 130, 153
default path for target file 132
RECEIVE.EXE location 32
used in sample program
BASIC 333
RECEIVE.EXE file 32
RECEIVE.EXE location 130
Release (12) 11, 133, 134
used in sample program
BASIC 333
Pascal 354
Reserve (11) 11, 133, 134
used in sample program
BASIC 330
Pascal 354
Reset System (21) 17, 22, 133, 134, 135, 151
used in sample program
BASIC 328
C 336
COBOL 344
Pascal 355
RESUME 155
return (output) parameters
general 51
processing 21
return codes
general 21
summary 30

S

sample code highlighting xix
Samples 327
Search Field (30) 136, 173
Search for String 221
search functions
Search Field (30) 136
Search Presentation Space (6) 138
Search Presentation Space (6) 8, 138, 173
used in sample program
BASIC 331
C 338, 340
Pascal 353
Send File (90) 11, 140, 153
default path for target file 142
SEND.EXE location 32, 140
used in sample program
C 339
Send Key (3) 33, 104, 143, 161, 173
used in sample program
BASIC 328
C 341
Pascal 353
Send Keystrokes 222
SEND.EXE file 32
sending keystrokes
mnemonics 32
Send Key (3) 143
service 122
Session Execute Macro 223
session keyboard 32
session options highlighting xix
session, host 25
Set Cursor (40) 150
used in sample program
BASIC 331
Set Cursor Position 230
Set Mouse Intercept Condition 231
Set Presentation Space Service Condition 234
Set Session Advise Condition 236
Set Session Parameters (9) 142, 143
general 151
List of affected functions 151
used in sample program
BASIC 332
C 336
COBOL 346
Valid Input 152—156
Set Structured Field Service Condition 238
shift key mnemonics 33
size of presentation space 19, 21, 25
declaring 25
determining 25
maximum number of 25
valid names 25
software requirements xx
source code syntax
BASIC (compiler) 38
BASIC (Interpretive) 39
C 41, 48
COBOL 42

source code syntax *(continued)*
Macro Assembler 43, 49
Pascal 45
specifying strings 9, 92
SRCHALL 152
SRCHBKWD 152
SRCHFROM 152
SRCHFRWD 152
Start Close Intercept 239
Start Close Intercept (41) 157
Start Host Notification (23) 31, 108, 117, 153, 159, 166, 169, 170
used in sample program
BASIC 328
C 336
COBOL 346
Start Keystroke Intercept 241
Start Keystroke Intercept (50) 161
used in sample program
COBOL 347
Start Menu Advise 276
Start Mouse Input Intercept 242
Start Playing Macro (110) 164
Start Read SF 246
Start Session Advise 249
statement, call
See Function calls
Stop Close Intercept 251
Stop Close Intercept (43) 165
Stop Host Notification (25) 166
used in sample program
BASIC 329
C 337
COBOL 346
Stop Keystroke Intercept 252
Stop Keystroke Intercept (53) 13, 167
used in sample program
COBOL 347
Stop Menu Advise 278
Stop Mouse Input Intercept 253
Stop Read SF 254
Stop Session Advise 255
Storage Manager (17) 31
functions associated with
Free All Storage subfunction 172
general 168
subfunctions
Free All Storage 172
Free Storage 170
Get Storage 169
Query Free Storage 171
used in sample program
COBOL 345, 348
STREOT 10, 152
string handling, Pascal 46
string interception, Get Key (51) 101
string specification 9, 152
STRLEN 10, 152
subdirectory, current 32
subfunctions 168
summary of EHLLAPI functions 53
swapping diskettes 32
syntax
See source code syntax

T

terminal emulator 3
Terminate Menu Conversation 279
Terminate Session Conversation 256
Terminate Structured Field Conversation 257
Terminate System Conversation 258
TIMEOUT 153
trace 15, 23, 154
TROFF 154
TRON 154
TWAIT 154, 173

U

utility services functions
Storage Manager (17) 168

V

variable names, highlighting xix

W

Wait (4) 8, 143, 173
used in sample program
BASIC 331
C 340
Pascal 353
Window Status (104) 174
Write SF 259
Write Structured Fields (127) 177

X

XLATE 155

IBM®

Part Number: 39H0428

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

39H 428

SC31-8266-