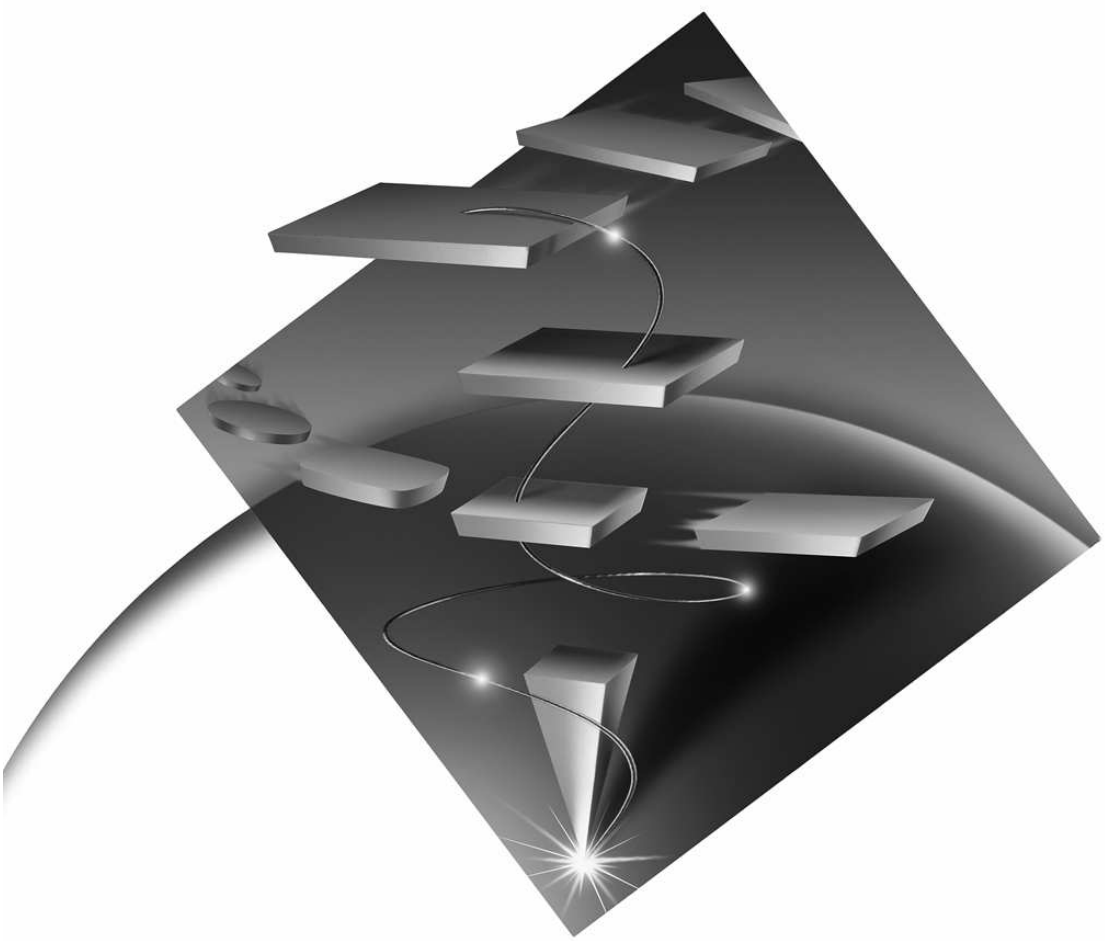
eNetwork Personal Communications
Version 4.2
for Windows 95 and Windows NT

Emulator Programming



SC31-8478-00

eNetwork Personal Communications
Version 4.2
for Windows 95 and Windows NT

Emulator Programming

SC31-8478-00

Note

Before using this information and the product it supports, be sure to read the general information under Appendix I, “Notices” on page 447.

First Edition (September 1997)

This edition applies to Version 4.2 of IBM eNetwork Personal Communications and to all subsequent releases and modifications until otherwise indicated in new editions or technical newsletters. Make sure you are using the correct edition for the level of the product.

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

IBM welcomes your comments. A form for readers' comments may be provided at the back of this publication, or you may address 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 1989, 1997. 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

About This Book xxi
Who Should Read This Book xxi
How This Book Is Organized xxi
Highlighting Conventions xxii
Related Publications xxiii
Notation xxiii

Chapter 1. Introduction to Emulator APIs 1
Cross Platform Support 1
Using API Header Files 2
Critical Sections 2
Stack Size 3
Sample Programs 3

Chapter 2. Introduction to EHLLAPI Programming 5
EHLLAPI Programming Overview 5
Languages 5
EHLLAPI Call Format 6
Data Structures 6
Memory Allocation 7
EHLLAPI Return Codes 7
Initialization 9
Compiling and Linking 9
Static Link Method 9
Dynamic Link Method 10
Presentation Spaces 10
32-bit Presentation Space IDs 10
Types of Presentation Spaces 11
Size of Presentation Spaces 11
Presentation Space Names 11
Host-Connected Presentation Space 11
Presentation Space ID Handling 12
Sharing EHLLAPI Presentation Space between Processes 14
ASCII Mnemonics 17
Debugging 19
A Simple EHLLAPI Sample Program 19
16/32 Bit Considerations 21
Host Automation Scenarios 22

Chapter 3. EHLLAPI Functions 29
Page Layout Conventions 29
Prerequisite Calls 29
Call Parameters 29
Return Parameters 29
Notes on Using This Function 29
Summary of EHLLAPI Functions 29
Allocate Communications Buffer (123) 31
Prerequisite Calls 31
Call Parameters 31
Return Parameters 32

© Copyright IBM Corp. 1989, 1997

Notes on Using This Function 32
Change PS Window Name (106) 32
Prerequisite Calls 32
Call Parameters 32
Return Parameters 33
Notes on Using This Function 33
Change Switch List LT Name (105) 34
Prerequisite Calls 34
Call Parameters 34
Return Parameters 34
Notes on Using This Function 35
Connect for Structured Fields (120) 35
Prerequisite Calls 35
Call Parameters 35
Return Parameters 36
Notes on Using This Function 36
Connect Window Services (101) 37
Prerequisite Calls 37
Call Parameters 37
Return Parameters 37
Notes on Using This Function 38
Connect Presentation Space (1) 38
Prerequisite Calls 38
Call Parameters 38
Return Parameters 39
Notes on Using This Function 39
Convert Position or Convert RowCol (99) 40
Prerequisite Calls 40
Call Parameters 40
Return Parameters 41
Notes on Using This Function 42
Copy Field to String (34) 42
Prerequisite Calls 43
Call Parameters 43
Return Parameters 43
Notes on Using This Function 44
Copy OIA (13) 47
Prerequisite Calls 48
Call Parameters 48
Return Parameters 48
Notes on Using This Function 48
Format of the Returned OIA Data String 49
Copy Presentation Space (5) 57
Prerequisite Calls 57
Call Parameters 57
Return Parameters 58
Notes on Using This Function 58
Copy Presentation Space to String (8) 62
Prerequisite Calls 62
Call Parameters 62
Return Parameters 63
Notes on Using This Function 63
Copy String to Field (33) 67
Prerequisite Calls 68

Call Parameters . . . 68
Return Parameters . . . 68
Notes on Using This Function . . . 68
Copy String to Presentation Space (15) . . . 70
Prerequisite Calls . . . 70
Call Parameters . . . 70
Return Parameters . . . 70
Notes on Using This Function . . . 71
Disconnect from Structured Fields (121) . . . 72
Prerequisite Calls . . . 72
Call Parameters . . . 72
Data String Contents . . . 73
Return Parameters . . . 73
Notes on Using This Function . . . 73
Disconnect Window Service (102) . . . 73
Prerequisite Calls . . . 74
Call Parameters . . . 74
Data String Contents . . . 74
Return Parameters . . . 74
Notes on Using This Function . . . 74
Disconnect Presentation Space (2) . . . 74
Prerequisite Calls . . . 75
Call Parameters . . . 75
Return Parameters . . . 75
Notes on Using This Function . . . 75
Find Field Length (32) . . . 75
Prerequisite Calls . . . 76
Call Parameters . . . 76
Return Parameters . . . 76
Notes on Using This Function . . . 77
Find Field Position (31) . . . 77
Prerequisite Calls . . . 77
Call Parameters . . . 77
Return Parameters . . . 78
Notes on Using This Function . . . 78
Free Communications Buffer (124) . . . 79
Prerequisite Calls . . . 79
Call Parameters . . . 79
Data String Contents . . . 79
Return Parameters . . . 79
Notes on Using This Function . . . 79
Get Key (51) . . . 80
Prerequisite Calls . . . 80
Call Parameters . . . 80
Data String Contents . . . 80
Return Parameters . . . 80
16-Bit Interface . . . 81
32-Bit Interface . . . 81
Notes on Using This Function . . . 82
Get Request Completion (125) . . . 83
Prerequisite Calls . . . 83
Call Parameters . . . 83
Data String Contents . . . 83
Return Parameters . . . 84

Notes on Using This Function . . . 85
Lock Presentation Space API (60) . . . 85
Prerequisite Calls . . . 86
Call Parameters . . . 86
Data String Contents . . . 86
Return Parameters . . . 86
Notes on Using This Function . . . 87
Lock Window Services API (61) . . . 87
Prerequisite Calls . . . 88
Call Parameters . . . 88
Data String Contents . . . 88
Return Parameters . . . 88
Notes on Using This Function . . . 89
Pause (18) . . . 89
Prerequisite Calls . . . 89
Call Parameters . . . 89
Return Parameters . . . 90
Notes on Using This Function . . . 90
Window Status (104) . . . 91
Prerequisite Calls . . . 91
Call Parameters . . . 91
Return Parameters . . . 94
Notes on Using This Function . . . 94
Post Intercept Status (52) . . . 94
Prerequisite Calls . . . 94
Call Parameters . . . 95
Return Parameters . . . 95
Query Close Intercept (42) . . . 95
Prerequisite Calls . . . 96
Call Parameters . . . 96
Return Parameters . . . 96
Query Communications Buffer Size (122) . . . 96
Prerequisite Calls . . . 97
Call Parameters . . . 97
Return Parameters . . . 97
Notes on Using This Function . . . 98
Query Cursor Location (7) . . . 98
Prerequisite Calls . . . 98
Call Parameters . . . 98
Return Parameters . . . 99
Query Field Attribute (14) . . . 99
Prerequisite Calls . . . 99
Call Parameters . . . 99
Return Parameters . . . 99
Notes on Using This Function . . . 100
Query Host Update (24) . . . 101
Prerequisite Calls . . . 101
Call Parameters . . . 101
Return Parameters . . . 102
Notes on Using This Function . . . 102
Query Window Coordinates (103) . . . 102
Prerequisite Calls . . . 102
Call Parameters . . . 102
Return Parameters . . . 103

Query Session Status (22) . . . 104
Prerequisite Calls . . . 104
Call Parameters . . . 104
Return Parameters . . . 104
Notes on Using This Function . . . 105
Query Sessions (10) . . . 105
Prerequisite Calls . . . 106
Call Parameters . . . 106
Return Parameters . . . 106
Notes on Using This Function . . . 107
Query System (20) . . . 107
Prerequisite Calls . . . 107
Call Parameters . . . 107
Return Parameters . . . 108
Return Code . . . 108
Notes on Using This Function . . . 109
Read Structured Fields (126) . . . 109
Prerequisite Calls . . . 109
Call Parameters . . . 109
Return Parameters . . . 110
Notes on Using This Function . . . 112
Receive File (91) . . . 114
Prerequisite Calls . . . 115
Call Parameters . . . 115
Return Parameters . . . 116
Notes on Using This Function . . . 117
Release (12) . . . 117
Prerequisite Calls . . . 117
Call Parameters . . . 117
Return Parameters . . . 117
Notes on Using This Function . . . 117
Reserve (11) . . . 118
Prerequisite Calls . . . 118
Call Parameters . . . 118
Return Parameters . . . 118
Notes on Using This Function . . . 118
Reset System (21) . . . 119
Prerequisite Calls . . . 119
Call Parameters . . . 119
Return Parameters . . . 119
Notes on Using this Function . . . 119
Search Field (30) . . . 119
Prerequisite Calls . . . 120
Call Parameters . . . 120
Return Parameters . . . 120
Notes on Using This Function . . . 121
Search Presentation Space (6) . . . 121
Prerequisite Calls . . . 122
Call Parameters . . . 122
Return Parameters . . . 122
Notes on Using This Function . . . 122
Send File (90) . . . 123
Prerequisite Calls . . . 124
Call Parameters . . . 124

Return Parameters . . . 125
Notes on Using This Function . . . 126
Send Key (3) . . . 126
Prerequisite Calls . . . 126
Call Parameters . . . 126
Return Parameters . . . 126
Notes on Using This Function . . . 127
Keyboard Mnemonics . . . 128
Set Cursor (40) . . . 135
Prerequisite Calls . . . 135
Call Parameters . . . 135
Return Parameters . . . 135
Notes on Using This Function . . . 135
Set Session Parameters (9) . . . 135
Prerequisite Calls . . . 136
Call Parameters . . . 136
Session Options . . . 137
Return Parameters . . . 146
Start Close Intercept (41) . . . 146
Prerequisite Calls . . . 146
Call Parameters . . . 146
Return Parameters . . . 147
Notes on Using This Function . . . 148
Start Host Notification (23) . . . 148
Prerequisite Calls . . . 148
Call Parameters . . . 148
Return Parameters . . . 150
Notes on Using This Function . . . 151
Start Keystroke Intercept (50) . . . 151
Prerequisite Calls . . . 151
Call Parameters . . . 151
Return Parameters . . . 153
Notes on Using This Function . . . 153
Start Playing Macro (110) . . . 154
Prerequisite Calls . . . 154
Call Parameters . . . 154
Return Parameters . . . 154
Stop Close Intercept (43) . . . 155
Prerequisite Calls . . . 155
Call Parameters . . . 155
Return Parameters . . . 155
Stop Host Notification (25) . . . 156
Prerequisite Calls . . . 156
Call Parameters . . . 156
Return Parameters . . . 156
Stop Keystroke Intercept (53) . . . 157
Prerequisite Calls . . . 157
Call Parameters . . . 157
Return Parameters . . . 157
Wait (4) . . . 157
Prerequisite Calls . . . 158
Call Parameters . . . 158
Return Parameters . . . 158
Notes on Using This Function . . . 158

Write Structured Fields (127) 158
Prerequisite Calls 159
Call Parameters 159
Return Parameters 160
Notes on Using This Function 161

Chapter 4. PCSAPI Functions 165
How to Use PCSAPI 165
Page Layout Conventions 165
Function Type 165
Parameter Type and Description 165
Return Code 165
pcsConnect Session 166
Function Type 166
Parameter Type and Description 166
Return Code 166
pcsDisconnect Session 166
Function Type 166
Parameter Type and Description 166
Return Code 166
pcsQueryEmulatorStatus 167
Function Type 167
Parameter Type and Description 167
Return Code 167
pcsQuerySessionList 167
Function Type 168
Parameter Type and Description 168
Return Parameters 168
Example 168
pcsQueryWorkstationProfile 169
Function Type 169
Parameter Type and Description 169
Return Code 169
pcsStartSession 170
Function Type 170
Parameter Type and Description 170
Return Code 170
pcsStopSession 170
Function Type 170
Parameter Type and Description 171
Return Code 171

Chapter 5. DDE Functions in a 32-Bit Environment 173
PC/3270 and PC400 DDE Data Items 173
Using System Topic Data Items 174
Using Session Topic Data Items 174
Using LU Topic Data Items (PC/3270 Only) 175
DDE Functions 175
Naming Conventions for Parameters 176
Find Field 176
Get Keystrokes 178
Get Mouse Input 179
Get Number of Close Requests 182
Get Operator Information Area 183

Get Partial Presentation Space 184
Get Presentation Space 186
Get Session Status 189
Get System Configuration 191
Get System Formats 192
PC/3270 or PC400 Response 192
Get System Status 192
PC/3270 or PC400 Response 193
Get System SysItems 193
PC/3270 or PC400 Response 194
Get System Topics 194
PC/3270 or PC400 Response 195
Get Trim Rectangle 195
PC/3270 or PC400 Response 195
Initiate Session Conversation 196
PC/3270 or PC400 Response 196
Initiate Structured Field Conversation 197
PC/3270 Response 197
Initiate System Conversation 197
PC/3270 or PC400 Response 198
Put Data to Presentation Space 198
PC/3270 or PC400 Response 198
Search for String 199
PC/3270 or PC400 Response 199
Structure of the Search Information 200
Send Keystrokes 200
PC/3270 or PC400 Response 201
Session Execute Macro 201
PC/3270 or PC400 Response 202
Issuing Commands with the Session Execute Macro Function 202
WINDOW Command 202
KEYBOARD Command 202
SEND Command 203
RECEIVE Command 203
SENDKEY Command 203
WAIT Command 207
Set Cursor Position 208
PC/3270 or PC400 Response 209
Set Mouse Intercept Condition 210
PC/3270 or PC400 Response 212
Set Presentation Space Service Condition 212
PC/3270 or PC400 Response 214
Set Session Advise Condition 214
PC/3270 or PC400 Response 215
Set Structured Field Service Condition 216
PC/3270 Response 216
Start Close Intercept 217
PC/3270 or PC400 Response 217
Start Keystroke Intercept 218
PC/3270 or PC400 Response 219
Start Mouse Input Intercept 219
PC/3270 or PC400 Response 220
Start Read SF 222
PC/3270 Response 223

Start Session Advise 224
PC/3270 or PC400 Response 225
Stop Close Intercept 225
PC/3270 or PC400 Response 226
Stop Keystroke Intercept 226
PC/3270 or PC400 Response 226
Stop Mouse Input Intercept 227
PC/3270 or PC400 Response 227
Stop Read SF 227
PC/3270 response 228
Stop Session Advise 228
PC/3270 or PC400 Response 228
Terminate Session Conversation 229
PC/3270 or PC400 Response 229
Terminate Structured Field Conversation 229
PC/3270 Response 229
Terminate System Conversation 229
PC/3270 or PC400 Response 230
Write SF 230
PC/3270 Response 231
DDE Menu Item API in a 32-Bit Environment 231
DDE Menu Client in a 32-Bit Environment 231
DDE Menu Server in a 32-Bit Environment 232
DDE Menu Functions in a 32-Bit Environment 233
Change Menu Item 233
Create Menu Item 241
Initiate Menu Conversation 242
Start Menu Advise 242
Stop Menu Advise 244
Terminate Menu Conversation 245
Summary of DDE Functions in a 32-Bit Environment 245

Chapter 6. Using DDE Functions with a DDE Client Application 251
Using the PC/3270 and PC400 DDE Interface 251
System Conversation 252
Session Conversation 253
Session Conversation (Hot Link) 254
PC/3270 and PC400 DDE Interface 255
DDE Functions for System Conversation 255
Get System Configuration 256
Get System Formats 256
Get System Status 256
Get System SysItems 257
Get System Topics 257
Initiate System Conversation 258
Terminate System Conversation 258
DDE Functions for Session Conversation 258
Find Field 258
Get Operator Information Area 259
Get Partial Presentation Space 260
Get Presentation Space 261
Get Session Status 261
Get Trim Rectangle 262
Initiate Session Conversation 263

Put Data to Presentation Space . . . 263
Search for String . . . 264
Session Execute Macro . . . 264
Set Cursor Position . . . 265
Terminate Session Conversation . . . 265
DDE Functions for Session Conversation (Hot Link) . . . 265
Initiate Session Conversation . . . 265
Start Close Intercept . . . 266
Start Keystroke Intercept . . . 266
Start Session Advise . . . 267
Stop Close Intercept . . . 268
Stop Keystroke Intercept . . . 269
Stop Session Advise . . . 269
Terminate Session Conversation . . . 269
Visual Basic Sample Program . . . 269

Chapter 7. Server-Requester Programming Interface (SRPI) Support . . . 283
How to Use SRPI . . . 283
SRPI Compatibility . . . 283
Using the Server-Requester Programming Interface . . . 284
SEND_REQUEST Parameters . . . 286
Supplied Parameters . . . 286
Returned Parameters . . . 288
How PC/3270 Applications Use SRPI . . . 288
Invoking SEND_REQUEST . . . 289
Performance Considerations . . . 289
Handling the Interrupt (Ctrl+Break) Key . . . 289
C Requesters . . . 289
C send_request Function . . . 290
SRPI Record Definition . . . 290
SRPI Return Codes . . . 290

Appendix A. Query Reply Data Structures Supported by EHLLAPI . . . 291
The DDM Query Reply . . . 291
DDM Application Name Self-Defining Parameter . . . 292
PCLK protocol controls Self-Defining Parameter . . . 292
Base DDM Query Reply Formats . . . 292
The IBM Auxiliary Device Query Reply . . . 294
Direct Access Self-Defining Parameter . . . 296
PCLK Protocol Controls Self-Defining Parameter . . . 296
The OEM Auxiliary Device Query Reply . . . 296
Direct Access Self-Defining Parameter . . . 297
PCLK Protocol Controls Self-Defining Parameter . . . 297
The Cooperative Processing Requester Query Reply . . . 298
The Product-Defined Query Reply . . . 299
Optional Parameters . . . 299
Direct Access Self-Defining Parameter . . . 300
The Document Interchange Architecture Query Reply . . . 301
Direct Access Self-Defining Parameter . . . 301

Appendix B. Compatibility with Personal Communications 4.1 for Windows 3.1 . . . 303

Appendix C. Compatibility with Communications Manager/2 . . . 305

Set Session Parameter (9) . . . 305
Set Options . . . 305
Return Parameters . . . 305
EAB Option . . . 305
Copy OIA (13) . . . 306
Copy String to PS (15) . . . 306
Storage Manager (17) . . . 307
Copy String to Field (33) . . . 307
Get Key (51) . . . 307
Window Status (104) . . . 307
Query Sessions (10) . . . 307
Connect for Structured Fields (120) . . . 308
Allocate Communications Buffer (123) . . . 308
ASCII Mnemonics . . . 308

Appendix D. DOS-Mode EHLLAPI . . . 309
Installation . . . 309
Compatibility . . . 309

Appendix E. SRPI Return Codes . . . 311
Error Handling . . . 311
Transport Layer Errors . . . 311
Application Errors . . . 311
SEND_REQUEST Processing Errors . . . 311
Types of SRPI Return Codes . . . 311
Type 0 Return Code Definitions . . . 312
Type 1 Return Code Definitions . . . 312
Type 2 Return Code Definitions . . . 314
Type 3 Return Code Definitions . . . 314
Class Definitions for Type 2 and Type 3 . . . 315
Exception Code Values for Type 2 and Type 3 . . . 315
Exception Object Values for Type 2 and Type 3 . . . 316
Server Return Codes . . . 317

Appendix F. DDE Functions in a 16-Bit Environment . . . 319
PC/3270 and PC400 DDE Data Items in a 16-Bit Environment . . . 319
Using System Topic Data Items . . . 320
Using Session Topic Data Items . . . 320
Using LU Topic Data Items (PC/3270 Only) . . . 321
DDE Functions in a 16-Bit Environment . . . 321
Naming Conventions for Parameters . . . 322
Find Field . . . 322
Get Keystrokes . . . 324
Get Mouse Input . . . 325
Get Number of Close Requests . . . 328
Get Operator Information Area . . . 329
Get Partial Presentation Space . . . 329
Get Presentation Space . . . 332
Get Session Status . . . 334
Get System Configuration . . . 336
Get System Formats . . . 337
PC/3270 or PC400 Response . . . 338
Get System Status . . . 338
PC/3270 or PC400 Response . . . 338

Get System SysItems . . . 339
PC/3270 or PC400 Response . . . 339
Get System Topics . . . 340
PC/3270 or PC400 Response . . . 340
Get Trim Rectangle . . . 341
PC/3270 or PC400 Response . . . 341
Initiate Session Conversation . . . 341
PC/3270 or PC400 Response . . . 342
Initiate Structured Field Conversation . . . 342
PC/3270 Response . . . 343
Initiate System Conversation . . . 343
PC/3270 or PC400 Response . . . 343
Put Data to Presentation Space . . . 343
PC/3270 or PC400 Response . . . 344
Search for String . . . 344
PC/3270 or PC400 Response . . . 345
Structure of the Search Information . . . 345
Send Keystrokes . . . 346
PC/3270 or PC400 Response . . . 346
Session Execute Macro . . . 347
PC/3270 or PC400 Response . . . 347
Issuing Commands with the Session Execute Macro Function . . . 347
WINDOW Command . . . 348
KEYBOARD Command . . . 348
SEND Command . . . 348
RECEIVE Command . . . 349
SENDKEY Command . . . 349
WAIT Command . . . 352
Set Cursor Position . . . 353
PC/3270 or PC400 Response . . . 354
Set Mouse Intercept Condition . . . 354
PC/3270 or PC400 Response . . . 356
Set Presentation Space Service Condition . . . 356
PC/3270 or PC400 Response . . . 357
Set Session Advise Condition . . . 358
PC/3270 or PC400 Response . . . 359
Set Structured Field Service Condition . . . 359
PC/3270 Response . . . 360
Start Close Intercept . . . 360
PC/3270 or PC400 Response . . . 361
Start Keystroke Intercept . . . 361
PC/3270 or PC400 Response . . . 362
Start Mouse Input Intercept . . . 363
PC/3270 or PC400 Response . . . 363
Start Read SF . . . 366
PC/3270 Response . . . 367
Start Session Advise . . . 367
PC/3270 or PC400 Response . . . 368
Stop Close Intercept . . . 369
PC/3270 or PC400 Response . . . 369
Stop Keystroke Intercept . . . 369
PC/3270 or PC400 Response . . . 370
Stop Mouse Input Intercept . . . 370
PC/3270 or PC400 Response . . . 370

Stop Read SF 371
PC/3270 Response 371
Stop Session Advise 371
PC/3270 or PC400 Response 372
Terminate Session Conversation 372
PC/3270 or PC400 Response 372
Terminate Structured Field Conversation 372
PC/3270 Response 373
Terminate System Conversation 373
PC/3270 or PC400 Response 373
Write SF 373
PC/3270 Response 374
Summary of DDE Functions in a 16-Bit Envionment 374
DDE Menu Item API in a 16-Bit Environment 378
DDE Menu Client in a 16-Bit Environment 378
DDE Menu Server, 32-Bit 379
DDE Menu Functions in a 16-bit Environment 380
Change Menu Item 381
Create Menu Item 387
Initiate Menu Conversation 388
Start Menu Advise 389
Stop Menu Advise 390
Terminate Menu Conversation 391

Appendix G. REXX EHLLAPI Functions 393
Overview of REXX EHLLAPI Function Calls and Return Values 393
Installation 393
Conventions 393
Summary of Prerequisite Calls for Functions 394
Summary of EHLLAPI and REXX EHLLAPI Functions 395
Change_Switch_Name 397
Change_Window_Name 398
Connect 399
Connect_PM 400
Convert_Pos 401
Copy_Field_To_Str 402
Copy_OIA 403
Copy_PS 404
Copy_PS_To_Str 405
Copy_Str_To_Field 406
Copy_Str_To_PS 407
Disconnect 408
Disconnect_PM 409
Find_Field_Len 410
Find_Field_Pos 411
Get_Key 412
Get_Window_Status 413
Intercept_Status 414
Lock_PMSVC 415
Lock_PS 416
Pause 417
Query_Close_Intercept 418
Query_Cursor_Pos 419
Query_Field_Attr 420

Query_Host_Update 421
Query_Session_Status 422
Query_Sessions 423
Query_System 424
Query_Window_Coord 425
Receive_File 426
ATTENTION 426
Release 427
Reserve 428
Reset_System 429
Search_Field 430
Search_PS 431
Send_File 432
ATTENTION 432
Sendkey 433
Set_Cursor_Pos 434
Set_Session_Parms 435
Set_Window_Status 436
Start_Close_Intercept 437
Start_Host_Notify 438
Start_Keystroke_Intercept 439
Stop_Close_Intercept 440
Stop_Host_Notify 441
Stop_Keystroke_Intercept 442
Wait 443
Programming Notes 443

Appendix H. Sample Programs 445

Appendix I. Notices 447

Appendix J. Trademarks 449

Glossary 451

Index 469

Figures

1. Keystroke Flow 26
2. Host Presentation Space Characters 50
3. DDE Menu Server Conversation 231
4. DDE Menu Client Conversation 232
5. Example of PC/3270 SRPI Requester and Server 284
6. IBM Workstation Requester and IBM Host Computer Server Relationship 285
7. Example of an SRPI Requester and Server Flow 286
8. DDE Menu Server Conversation 379
9. DDE Menu Client Conversation 380

Tables

1. Emulator API Support 2
2. EHLLAPI Read and Write Sharing Option Combinations 15
3. Prerequisite Functions and Associated Dependent Functions 16
4. Call Parameters 31
5. Calling Data String 31
6. Call Parameters 32
7. Calling Data String 33
8. Call Parameters 34
9. Calling Data String 34
10. Call Parameters 35
11. Calling Data String 35
12. Call Parameters 37
13. Calling Data String 37
14. Call Parameters 38
15. Calling Data String 39
16. Call Parameters 40
17. Calling Data String 41
18. Call Parameters 43
19. Call Parameters 48
20. Calling Data String 49
21. Call Parameters 57
22. Call Parameters 62
23. Call Parameters 68
24. Call Parameters 70
25. Call Parameters 72
26. Calling Data String 73
27. Call Parameters 74
28. Calling Data String 74
29. Call Parameters 75
30. Call Parameters 76
31. Call Parameters 77
32. Call Parameters 79
33. Calling Data String 79
34. Call Parameters 80
35. Calling Data String 80

© Copyright IBM Corp. 1989, 1997

36. Calling Data String 80
37. Call Parameters 83
38. Calling Data String 83
39. Calling Data String 84
40. Call Parameters 86
41. Calling Data String 86
42. Call Parameters 88
43. Calling Data String 88
44. Call Parameters 89
45. Call Parameters 91
46. Calling Data String 91
47. Calling Data String 91
48. Calling Data String 92
49. Calling Data String 93
50. Call Parameters 95
51. Calling Data String 95
52. Call Parameters 96
53. Calling Data String 96
54. Call Parameters 97
55. Calling Data String 97
56. Call Parameters 98
57. Call Parameters 99
58. Call Parameters 101
59. Calling Data String 102
60. Call Parameters 102
61. Calling Data String 103
62. Calling Data String 103
63. Call Parameters 104
64. Calling Data String 104
65. Call Parameters 106
66. Calling Data String 106
67. Call Parameters 107
68. Calling Data String 108
69. Call Parameters 109
70. Calling Data String 110
71. Call Parameters 110
72. Return Parameters 111
73. Call Parameters 115
74. Call Parameters 117
75. Call Parameters 118
76. Call Parameters 119
77. Call Parameters 120
78. Call Parameters 122
79. Call Parameters 124
80. Call Parameters 126
81. Mnemonics with Uppercase Alphabetic Characters 128
82. Mnemonics with Numbers or Lowercase Characters 129
83. Mnemonics with @A and @ Uppercase Alphabetic Characters 129
84. Mnemonics with @A and @ Lowercase Alphabetic Characters 130
85. Mnemonics with @A and @ Alphanumeric (Special) Characters 131
86. Mnemonics with @S (Shift) and @ Alphabetic Characters 131
87. Mnemonics Using @X and @Alphabetic Lowercase (For DBCS Only) 131
88. Mnemonics Using @M, @Q and @Alphabetic Lowercase (For VT Only) 131
89. Mnemonics with Special Character Keys 134

90. Call Parameters . . . 135
91. Call Parameters . . . 136
92. Call Parameters . . . 146
93. Calling Data String . . . 146
94. Calling Data String . . . 147
95. Calling Data String . . . 147
96. Call Parameters . . . 148
97. Calling Data String . . . 149
98. Returned Data String . . . 150
99. Returned Data String . . . 150
100. Call Parameters . . . 151
101. Calling Data String . . . 152
102. Returned Data String . . . 152
103. Call Parameters . . . 154
104. Calling Data String . . . 154
105. Call Parameters . . . 155
106. Calling Data String . . . 155
107. Call Parameters . . . 156
108. Calling Data String . . . 156
109. Call Parameters . . . 157
110. Calling Data String . . . 157
111. Call Parameters . . . 158
112. Call Parameters . . . 159
113. Calling Data String . . . 159
114. Return Parameters . . . 160
115. Return Parameters . . . 161
116. Naming Scheme for Data Items . . . 173
117. Applications for PC/3270 and PC400 . . . 173
118. Topics for PC/3270 and PC400 . . . 174
119. SENDKEY Command List . . . 204
120. DDE Function Summary in a 32-Bit Environment . . . 245
121. Naming Scheme for Data Items . . . 251
122. Topics for PC/3270 or PC400 . . . 251
123. Parameters Supplied by the SRPI Requester . . . 287
124. Parameters Returned to the SRPI Requester . . . 288
125. DDM Query Reply Base Format . . . 291
126. DDM Application Name Self-Defining Parameter . . . 292
127. DDM PCLK Auxiliary Device Self-Defining Parameter . . . 292
128. Base DDM Query Reply Format with Name and Direct Access Self-Defining Parameters . . . 293
129. Base DDM Query Reply Format with Direct Access and Name Self-Defining Parameters . . . 294
130. IBM Auxiliary Device Base Format with Direct Access Self-Defining Parameter . . . 295
131. IBM Auxiliary Device Direct Access Self-Defining Parameter . . . 296
132. IBM Auxiliary Device PCLK Self-Defining Parameter . . . 296
133. OEM Auxiliary Device Base Format with Direct Access Self-Defining Parameter . . . 297
134. OEM Auxiliary Device Direct Access Self-Defining Parameter . . . 297
135. IBM Auxiliary Device PCLK Self-Defining Parameter . . . 298
136. CPR Query Reply Buffer Format . . . 298
137. IBM Product-Defined Query Reply Base Format . . . 299
138. Valid REFID and SSID Values for the IBM Product-Defined Query Reply 300
139. IBM Product-Defined Direct Access Self-Defining Parameter . . . 300

140. IBM DIA Base Format 301
141. IBM Product-Defined Direct Access Self-Defining Parameter 301
142. Type 1 Return Code Definitions and Descriptions 312
143. Type 3 Return Code Definitions and Descriptions 315
144. Class Definitions for Type 2 and Type 3 315
145. Exception Code Values for Type 2 and Type 3 316
146. Exception Object Values for Type 2 and Type 3 316
147. Naming Scheme for Data Items 319
148. Applications for PC/3270 and PC400 319
149. Topics for PC/3270 and PC400 320
150. SENDKEY Command List 349
151. Summary of DDE Functions in a 16-Bit Environment 374
152. Prerequisite Calls for Functions 394
153. EHLLAPI and REXX EHLLAPI Functions 395

About This Book

This book provides necessary programming information for you to use the IBM Personal Communications /3270 and the IBM Personal Communications AS/400 for Windows 95 and Windows NT (hereafter called *PC/3270* and *PC400*, respectively) Emulator High-Level Language Application Program Interface (EHLLAPI), Dynamic Data Exchange (DDE), Personal Communications API (PCSAPI), and Server-Requester Programming Interface (SRPI).

EHLLAPI/DDE/PCSAPI 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 workstation session.

Note: SRPI is supported by PC/3270 only.

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 Version 4.2 Quick Beginnings for Windows 95 and Windows NT.*

Who Should Read This Book

This book is intended for workstation users and programmers who write application programs that use functions.

A working knowledge of Windows 95 and Windows NT is assumed. For information about Windows 95 and Windows NT, refer to the list of publications under “Related Publications” on page xxiii.

The programmer must also be familiar with connecting to a host system from a terminal or from a workstation 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 for the appropriate references for the specific language you are using.

How This Book Is Organized

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

Chapter 2, Introduction to EHLLAPI Programming, contains information about migrating from previous versions of Personal Communications to Personal Communications Version 4.2 for Windows 95 and Windows NT.

Chapter 3, EHLLAPI Functions, describes each PC/3270 and PC400 EHLLAPI function in detail. The functions are arranged alphabetically by name. See page 245 when you want to find the EHLLAPI function by the function number.

Chapter 4, PCSAPI Functions, describes PCSAPI function supported in the PC/3270 Windows mode and PC400.

Chapter 5, DDE Functions in a 32-Bit Environment, provides information concerning Dynamic Data Exchange functions of PC/3270 and PC400.

© Copyright IBM Corp. 1989, 1997

Chapter 6, Using DDE Functions with a DDE Client Application, describes how to use DDE functions with other Windows** application programs (Microsoft** Visual Basic**, Microsoft Excel**, Microsoft Word**) that have DDE client functions.

Chapter 7, Server-Requester Programming Interface (SRPI) Support, includes information about PC/3270 for Windows 95 and Windows NT support for SRPI functions.

Appendix A, Query Reply Data Structures Supported by EHLLAPI, includes reference information about the Query Reply data structures supported by EHLLAPI.

Appendix B, Compatibility with Personal Communications 4.1 for Windows 3.1, includes reference information about Personal Communications V3.1 functions.

Appendix C, Compatibility with Communications Manager/2, describes the compatibility with Communications Manager/2.

Appendix E, SRPI Return Codes, describes error handling in the SRPI environment.

Appendix F, DDE Functions in a 16-Bit Environment, describes 16-bit DDE functions.

Appendix G, REXX EHLLAPI Functions, describes REXX EHLLAPI functions.

Appendix H, Sample Programs, tells where to find REXX EHLLAPI sample programs.

Highlighting Conventions

Highlighting refers to different type styles used within text to show 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 for.

Type style	Emphasized Information
ROMAN CAPS	ROMAN CAPS are used for commands, keywords, or other information that the programmer is to type in literally. This type style is not used in lengthy sample program listings.
Bold	A **Bold** type style (Initial Caps) is used for EHLLAPI function names. Keep in mind that these names are labels used to refer to the function. They are not used *in* an application to *call* an EHLLAPI function; the *function number* parameter is used to specify which EHLLAPI function is to be performed. See Chapter 3, 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.
Bold Monospace	Bold monospace type indicates file and directory names. It is also used for programmable keywords.

Italic Monospace

Italic monospace type indicates variables for placeholders, parameters, and file names. When you enter a command, replace all variables with the appropriate file name, number, or option. On an example display screen, nonitalic regular monospace type indicates a variable quantity other than a default value.

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 135.

For ASCII mnemonics in text.

For displayed information.

For input from either a programmer or an operator.

Related Publications

Following is a list of related publications:

Personal Communications Version 4.2 Quick Beginnings for Windows 95 and Windows NT, SC31-8476

Personal Communications Version 4.2 Reference for Windows 95 and Windows NT, SC31-8477

Personal Communications Version 4.2 System Management Programming for Windows 95 and Windows NT, SC31-8480

Personal Communications Version 4.2 Client/Server Programming for Windows 95 and Windows NT, SC31-8479

IBM 3270 Information Display System Data Stream Programmer's Reference, GA23-0059

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 3, EHLLAPI Functions, Chapter 4, PCSAPI Functions, Chapter 5, DDE Functions in a 32-Bit Environment, and Appendix F, DDE Functions in a 16-Bit Environment. It shows the products that provide the function described in the section. 'Yes' means it is supported, and 'No' means not supported. For example, the following table indicates that a function is available in the PC/3270 Windows mode and it is not supported in the PC400.

3270	*5250*	*VT*
Yes	No	Yes

Chapter 1. Introduction to Emulator APIs

The IBM eNetwork Personal Communications product supplies several application programming interfaces (APIs). Each interface has a specific set of functions and may be used for different purposes. Choose the programming interface that best matches the functional requirements of your application. Some applications may use more than one interface to achieve the desired results. The programming interfaces are:

Emulator High Level Language API (EHLLAPI): This interface provides functions to access emulator "presentation space" data such as characters on the host screen. It also provides functions for sending keystrokes to the host, intercepting user-entered keystrokes, querying the status of the host session, uploading and downloading files, and other functions. This interface is often used for 'automated operator' applications which read host screens and enter keystrokes without direct user intervention. See Chapter 3, “EHLLAPI Functions” on page 29.

Dynamic Data Exchange (DDE): This interface is similar to the EHLLAPI interface in that it provides a programmable means to read the host screen, send keystrokes, and perform related functions. It has some additional functions for access to the emulator clipping rectangle, intercepting mouse events, and adding/removing commands on the emulator menu bar. See Chapter 5, “DDE Functions in a 32-Bit Environment” on page 173.

Personal Communications Session API (PCSAPI): This interface is used to start, stop, and control emulator sessions. See Chapter 5, “DDE Functions in a 32-Bit Environment” on page 173.

Server-Requestor Programming Interface (SRPI): This interface is used in cooperation with an IBM Enhanced Connectivity Facility (ECF) application running on a host system. This API provides functions for writing synchronous call-return interfaces to remote server programs. See Chapter 7, “Server-Requester Programming Interface (SRPI) Support” on page 283.

IBM eNetwork Personal Communications Host Access Class Library (ECL): ECL is a set of objects that allow application programmers and scripting language writers to access host applications easily and quickly. Personal Communications supports three different ECL layers (C++ objects, ActiveAutomation (OLE), and LotusScript Extension (LSX)). See *Personal Communications Version 4.2 Host Access Class Library* for more details.

Cross Platform Support

The emulator programming interfaces are provided across a variety of workstation platforms. On some platforms 16- and 32-bit interfaces may be provided. Note that support for emulator APIs on a particular platform requires a licensed copy of IBM Personal Communications for that platform.

The following table shows the emulator API support on various platforms:

© Copyright IBM Corp. 1989, 1997

Table 1. Emulator API Support

Supported PCOM Emulator APIs	PCOM DOS	PCOM OS/2	PCOM Windows 3.1	PCOM Windows 95 & NT
EHLLAPI (16-bit)	Yes	Yes	Yes	Yes
EHLLAPI (32-bit)		Yes		Yes
EHLLAPI (DOS-mode) *		Yes	Yes	Yes
EHLLAPI (WinOS/2-mode) **		Yes		Yes
PCSAPI (16-bit)			Yes	
PCSAPI (32-bit)		Yes		Yes
DDE (16-bit)		Yes	Yes	Yes
DDE (32-bit)		Yes		Yes
DDE (WinOS/2-mode) **		Yes		
SRPI (16-bit)		Yes	Yes	
SRPI (32-bit)		Yes		Yes
Host Access Class Library				Yes

Using API Header Files

The application program should include operating system header files before including API header files. For example:

```
#include <windows.h>          // Windows main header
#include "pcsapi.h"           // PComm PCSAPI header
...
```

Critical Sections

Use critical sections (Windows **EnterCriticalSection** function) carefully when your program calls emulator APIs. Do not make emulator API calls within a critical section. If one thread of an application establishes a critical section and another thread is within an emulator API call, the call is suspended until the critical section is exited.

During processing of an API call, all signals (except numeric coprocessor signals) are delayed until the call completes or until the call needs to wait for incoming data. Also, **TerminateProcess** issued from another process is held until the application completes an API call it might be processing.

* DOS-mode is support for DOS applications in a DOS-compatibility mode of the operating system; for example the DOS emulation window of OS/2.

** WinOS2-mode is support for 16-bit Windows applications in a Win-OS2 session on OS/2 using the Personal Communications for OS/2 product.

Stack Size

Emulator APIs use the calling programs stack when they are executed. The operating system, the application, and the API all require stack space for dynamic variables and function parameters. At least 8196 bytes (8K) of stack space should be available at the time of an API call. It is the responsibility of the application program to insure sufficient stack space is available for the API.

Sample Programs

Several sample programs are provided, each of which illustrates the use of one of the Personal Communications APIs. If you choose to install the sample programs, they will be installed in the default directory: /SAMPLES.

Note: International Business Machines Corporation provides these files "as is," without warranty of any kind, either express or implied, including, but not limited to, the implied warranties of merchantability or fitness for a particular purpose.

The sample programs provided with Personal Communications V4.2 are located on both diskette and CD-ROM media. The sample program files include source and supporting files for the following Personal Communications Version 4.2 for Windows 95 and Windows NT APIs:

Emulator High-Level Language Programming Interface (EHLLAPI)
Dynamic Data Exchange (DDE)
Server-Requester Programming Interface (SRPI)
PCSAPI Functions

The following files contain information about the sample programs in a subdirectory of that name:

File Name	Description
DDXFER.DOC	Shows how EHLLAPI can be used to create a “Drag and Drop” application; in this case, for file transfer
HLLSMP.DOC	Shows how to use EHLLAPI to request a keystroke and log on to a VM system
LISTFILE.DOC	Illustrates how DDE can make use of the LOAD button to transfer files from the host
PCSMAIN.DOC	Illustrates the use of PCSAPI to start and stop sessions, query the session status, and query the profile for the session.
SPL2FILE.DOC	Is a program that uses DDE to save an AS/400 spool file as an ASCII file on the PC
SRPSMP.DOC	Illustrates the use of the Server Requester Programming Interface (SRPI)

The following files will be installed in the \SAMPLES directory:

File Name	Description
DDE_C.H	DDE include file
HAPI_C.H	EHLLAPI include file

PCSAPI.H	PCSAPI include file
PCSCALLS.LIB	Import library for 16-bit interface
PCSCAL32.LIB	Import library for 32-bit interface
UUCCPRB.H	SRPI include file

The following subdirectories will be created in the \SAMPLES directory:

Subdirectory	Description
DDXFER	DDXFER sample files
ELC	ELC sample files
HLLSMP	EHLLAPI sample files
LISTFILE	LISTFILE sample files
PCSMAIN	PCSAPI sample files
SPL2FILE	SPL2FILE sample files
SRPSMP	SRPI sample files
VBDDE	VBDDE sample files
VBHLLAPI	VBHLLAPI sample files
VBPCSAPI	VBPCSAPI sample files

Chapter 2. Introduction to EHLLAPI Programming

This chapter provides information needed to incorporate EHLLAPI functions into applications written in a high level language. It provides details on the EHLLAPI call format, memory allocation considerations, how to initialize the EHLLAPI interface, and how to compile and link EHLLAPI applications. Also included is a short sample EHLLAPI program and the compile/link instructions used to build it. Finally, a set of possible uses for the EHLLAPI interface (scenarios) is described.

An "EHLLAPI application" is any application program which uses the EHLLAPI interface to access the host 3270/5250/VT presentation space. The presentation space includes the visible emulator character data, fields and attribute data, keystroke data, and other information.

EHLLAPI Programming Overview

EHLLAPI is a programming interface which allows programmatic access to a host emulator session. Functions are provided for reading host screen data (such as the characters and attributes), for sending keystrokes, and performing other emulator-related functions.

The EHLLAPI interface is a single call-point interface. There is a single callable API through which all EHLLAPI functions are requested. On each call to the interface the application provides a function number which identifies the function requested, a pointer to a data buffer, a pointer to the length of the data buffer, and a pointer to a return code (see “EHLLAPI Call Format” on page 6).

Languages

Any programming language which can invoke an entry point in a DLL with the "pascal" calling convention can be used to execute EHLLAPI functions. However, the Personal Communications EHLLAPI toolkit provides header files and function prototypes only for the C/C++ languages. A clear understanding of data structure layout and calling conventions is required to use any other language. The EHLLAPI toolkit supports the following C/C++ compilers:

IBM Visual Age for C++ for Windows Version 3.5

Microsoft Visual C++ Version 4.0

Most other C/C++ compilers will also work with the toolkit.

EHLLAPI C/C++ applications must include the Personal Communications EHLLAPI header file (HAPI_C.H). This file defines the layout of data structures and provides a prototype for the EHLLAPI entry point.

Note: The data structure layout for 16 and 32 bit applications are not the same (see “16/32 Bit Considerations” on page 21).

© Copyright IBM Corp. 1989, 1997

EHLLAPI Call Format

The EHLLAPI entry point (**hllapi**) is always called with the following four parameters:

1. EHLLAPI Function Number (input)
2. Data Buffer (input/output)
3. Buffer Length (input/output)
4. Presentation Space Position (input); Return Code (output)

Each parameter is passed by *reference* not by value. Thus each parameter to the function call must be a *pointer* to the value, not the value itself. For example, the following is a correct example of calling the EHLLAPI Query Session Status function:

```
#include  "hapi_c.h"
 struct HLDQuerySessionStatus QueryData;
int      Func,  Len,  Rc;

 memset(QueryData, 0, sizeof(QueryData)); // Init buffer
 QueryData.qsst_shortname = 'A';                  // Session to query
 Func = HA_QUERY_SESSION_STATUS;                    // Function number
 Len    = sizeof(QueryData);                        // Len of buffer
 Rc     = 0;                                           // Unused on input

 hllapi(&Func, (char *)&QueryData, &Len, &Rc);    // Call EHLLAPI
 if (Rc != 0) {                                        // Check return code
    // ...Error handling
}
```

All the parameters in the **hllapi** call are pointers and the return code of the EHLLAPI function is returned in the value of the 4th parameter, not as the value of the function. For example, the following is **not** correct:

```
 if (hllapi(&Func, (char *)&QueryData, &Len, &Rc) != 0) { // WRONG!
    // ...Error handling
}
```

Although the **hllapi** function is defined to return an **int** data type, its value is undefined and should not be used.

The second through fourth parameters of the **hllapi** call can return information to the application. The description of each EHLLAPI function describes what, if any, information is returned in these parameters.

Data Structures

Many EHLLAPI functions use a formatted data structure to pass information to or from the application program. The description of each function shows the layout of the data structure. The data passed to or from the EHLLAPI function must exist in storage exactly as documented, byte for byte. Note that the structure layout is different for 16 and 32 bit applications. EHLLAPI data structures are packed to a byte alignment.

It is *highly recommended* that the supplied header file and data structure definitions be used to insure proper data alignment and layout. Although it is technically possible, the following is *not* recommended:

```
char QueryData[20];    // Not recommended
...
Func = HA_QUERY_SESSION_STATUS;
hllapi(&Func, QueryData, &Len, &Rc);
if (QueryData[13] == 'F') {
   // ...this is a 5250 sessions
}
```

The recommended way to write this function would be:

```
#include  "hapi_c.h"
struct HLDQuerySessionStatus QueryData;    // Recommended
...
Func = HA_QUERY_SESSION_STATUS;
hllapi(&Func, (char *)&QueryData, &Len, &Rc);
if (QueryData.qsst_sestype == 'F') {
   // ...this is a 5250 sessions
}
```

Memory Allocation

EHLLAPI functions do not allocate or free memory. The application program must preallocate buffer space for EHLLAPI functions which require it before calling the **hllapi** entry point. The buffer space may be preallocated as a dynamic variable such as:

```
struct HLDQuerySessionStatus QueryBuff;
```

or it may be allocated by a call to a C library or operating system function such as:

```
struct HLDQuerySessionStatus *QueryBuff;
...
QueryBuff = malloc(sizeof(struct HLDQuerySessionStatus));
```

In any case, the application is responsible for allocating sufficient buffer space before calling EHLLAPI functions and for freeing buffers when they are not needed.

EHLLAPI Return Codes

EHLLAPI functions return a completion code in the 4th parameter of the **hllapi** function call (except for the **Convert Position** or **RowCol** (99) function). The return code indicates the success or failure of the requested function.

Unless indicated otherwise in the description of each function, the following table shows the meaning of each return code value. Some functions may have a slightly different interpretation of these return codes; refer to the individual function descriptions for details.

Return Code	Explanation
0	The function successfully executed, or no update since the last call was issued.

Return Code	Explanation
1	An incorrect host presentation space ID was specified. The 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 number was specified. (Refer to the 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 not valid.
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.
12	This session stopped.
24	The string was not found, or the presentation space is unformatted.
25	Keystrokes were not available on input queue.
26	A host event occurred. See **Query Host Update** (24) for details.
27	File transfer was ended by a Ctrl+Break command.
28	Field length was 0.
31	Keystroke queue overflow. Keystrokes were lost.
32	An application has already connected to this session for communications.
33	Reserved.
34	The message sent to the host was canceled.
35	The message sent from the host was canceled.
36	Contact with the host was lost.
37	Inbound communication has been disabled.
38	The requested function has not completed its execution.
39	Another DDM session is already connected.
40	The disconnection attempt was successful, but there were asynchronous requests that had not been completed at the time of the disconnection.
41	The buffer you requested is being used by another application.
42	There are no outstanding requests that match.
43	The API was already locked by another EHLLAPI application (on LOCK) or API not locked (on UNLOCK).

Initialization

An application which uses EHLLAPI functions should first call the **Reset System (21)** function to set all EHLLAPI parameters to the default conditions. The **Query Sessions (10)** can then be used to discover what host sessions are available.

Compiling and Linking

Applications using EHLLAPI functions must include the supplied header file, **hapi_c.h**, to obtain the proper function prototypes, constants, and data structure definitions. This header file may be used with 16 or 32 bit programs compiled with any of the supported C/C++ compilers (see “Languages” on page 5). If a different compiler or language is used, then you must provide your own equivalent definitions and structures.

There are two possible ways to link the application program, depending on how the **hllapi** entry point is to be resolved. The simplest way is to statically link the application with the Personal Communications **PCSCALLS.LIB** (16-bit) or **PCSCAL32.LIB** (32-bit) library. This will resolve the entry point at link time. The operating system will load the EHLLAPI DLL with the application when it starts. Another way to link to the **hllapi** entry point is to perform dynamic linking. In this case the application uses operating system calls to load the EHLLAPI DLL and obtain the **hllapi** entry point address at run time.

Static Link Method

Using the static link method the application can simply call the **hllapi** entry point when needed such as:

```
#include  "hapi_c.h"
 int HFunc, HLen, HRc;                                  // Function parameters
 char HBuff[1];                                         // Function parameters
 ...
 HFunc = HA_RESET_SYSTEM;                                     // Run EHLLAPI function
HLen   = 0;
HRc    = 0;
 hllapi(&Func, HBuff, &HLen, &HRc);
 if (HRc != 0) {
    // ... EHLLAPI access error
 }
```

When the application is linked, the appropriate Personal Communications library files (**PCSCALLS.LIB** for 16-bit or **PCSCAL32.LIB** for 32-bit) must be linked with the application executable code. For example, the following link command might be used (IBM Visual Age C/C++):

```
ilink /de /noe pcscal32.lib sample.obj
```

When the operating system loads an application constructed in this way, the Personal Communications EHLLAPI module is loaded automatically.

Dynamic Link Method

Using the dynamic link method the application makes calls to the operating system at run time to load the Personal Communications EHLLAPI module and to locate the **hllapi** entry point within it. This method requires more code in the application but gives the application greater control over error conditions. For example, the application can display a specific error message to the user if the Personal Communications EHLLAPI module cannot be found.

To use dynamic linking, the application needs to load the PCSHLL.DLL (16-bit) or PCSHLL32.DLL (32-bit) Personal Communications module and locate the **hllapi** entry point. It is recommended that the entry point be located by its ordinal number and not by name. The ordinal number is defined in the **hapi_c.h** header file. The following 32-bit Windows code loads the 32-bit EHLLAPI module, locates the **hllapi** entry point, and makes an EHLLAPI function call.

```
#include  "hapi_c.h"

 HMODULE Hmod;                                                  // Handle of PCSHLL32.DLL
 long (APIENTRY hllapi)(int *, char *, int *, int *);    // Function pointer
 int HFunc, HLen, HRc;                                          // Function parameters
 char HBuff[1];                                                 // Function parameters

 Hmod = LoadLibrary("PCSHLL32.DLL");                            // Load EHLLAPI module
 if (Hmod == NULL) {
    // ... Error, cannot load EHLLAPI module
 }

 hllapi = GetProcAddress(Hmod, MAKEINTRESOURCE(ord_hllapi));
                                                                // Get EHLLAPI entry point
 if (hllapi == NULL) {
    // ... Error, cannot find EHLLAPI entry point
 }

 HFunc = HA_RESET_SYSTEM;                                       // Run EHLLAPI function
HLen   =  0;
HRc     =  0;
 (*hllapi)(&Func, HBuff, &HLen, &HRc);
 if (HRc != 0) {
    // ... EHLLAPI access error
 }
```

Presentation Spaces

Many EHLLAPI functions require a *presentation space ID (PSID)* to indicate which host emulator session is to be used for the function. (This is also referred to as the *short session ID*). A presentation space ID is a single character in the range 'A' to 'Z'. There are a maximum of 26 sessions.

32-bit Presentation Space IDs

For 32-bit EHLLAPI applications the session ID is extended with 3 additional bytes. These extended session bytes must be set to zero for future compatibility. This is most easily accomplished by setting the contents of EHLLAPI buffers to all binary zero before filling them in with the required information. For example, the following might be used to query the status of session 'B':

```
#include  "hapi_c.h"
 int HFunc, HLen, HRc;                                    // Function parameters
 struct HLDPMWindowStatus StatusData;                     // Function parameters

 Func = HA_PM_WINDOW_STATUS;
 HLen = sizeof(StatusData);
HRc   =  0;

 // Set data buffer to zeros and fill in request
 memset(&StatusData, 0x00, sizeof(StatusData));
 StatusData.cwin_shortname = 'B';          // Short session ID
 StatusData.cwin_option       = 0x02;       // Query command

 hllapi(&Func, (char *)&StatusData, &HLen, &HRc);
```

Types of Presentation Spaces

An emulator session can be configured as a display session or a printer session. EHLLAPI applications cannot connect to printer or router sessions of PC400. The **Query Sessions (10)** function can be used to determine the type of a particular session.

Size of Presentation Spaces

An emulator display session can be configured for a range of screen sizes from 1920 bytes (24x80 screen size) to 9920 bytes (62x160 screen size). Some EHLLAPI functions such as **Copy PS to String (8)** require the application to allocate enough storage to hold (possibly) the entire presentation space. The size of the presentation space for a given session can be obtained using the **Query Session Status (22)** function.

Presentation Space Names

PC/3270 or PC400 EHLLAPI supports up to 26 presentation spaces or sessions. Valid names are the letters *A* through *Z*. To be valid within an application, a name must be declared during customization. See the *Personal Communications Version 4.2 Reference for Windows 95 and Windows NT* or the *Personal Communications Version 4.2 Quick Beginnings for Windows 95 and Windows NT.*

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) is used to identify 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 by using the **Connect Presentation Space** (1) and **Disconnect Presentation Space** (2) functions. The status of the connection determines whether some functions can be executed. It also affects how the PSID is defined. The following text explains how to control the status of the connection to the host presentation space:

At any given time, there can be either no host-connected presentation space, or there can be one and only one host-connected presentation space.

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

Following a connect, there is one and only one host-connected presentation space. The host presentation space that is connected 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 one and only one host-connected presentation space. Again, the host presentation space that is connected 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 following multiple consecutive calls to connect or following a single call to connect.

You cannot connect to a logical printer session.

Presentation Space ID 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 two factors:

1. The method used to specify the PSID:
   a. As the calling data string parameter of a preceding call to the **Connect Presentation Space** (1) function
   b. As a character in the calling data string of the function being executed. Handling varies depending on whether the character is:

      A letter *A* through *Z*
      A blank or a null
2. 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. 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 any 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. They are as follows:

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* through *Z*
A blank or a null

The first two functions require that a letter be used to specify the 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, is used to specify the PSID.
2. If a blank or a null is used to specify the PSID, a return code of 1 is generated. The function does not execute.
3. Using a letter to specify the 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 the 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 is used to specify the PSID.
2. If a blank or a null is used to specify the 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 the PSID does not change the established PSID of the host-connected presentation space. Unless another connect is executed:

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

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

Sharing EHLLAPI Presentation Space between Processes

More than one EHLLAPI application can share a presentation space if the applications support sharing (that is, if they were developed to work together or if they exhibit predictable behavior). To determine which applications support sharing, EHLLAPI applications are specified as one of following types:

Supervisory
Exclusive write with read privilege allowed
Exclusive write without read privilege allowed
Super write
Read

The type of shared access can be defined by setting the following read and write sharing options for each function in the **Set Session Parameters** (9) function call:

SUPER_WRITE

The application allows other applications that allow sharing and have write access permissions to concurrently connect to the same presentation space. The originating application performs supervisory-type functions but does not create errors for other applications that share the presentation space.

WRITE_SUPER

The application requires write access and allows only supervisory applications to concurrently connect to its presentation space. This is the default value.

WRITE_WRITE

The application requires write access and allows partner or other applications with predictable behavior to share the presentation space.

WRITE_READ

The application requires write access and allows other applications that perform read-only functions to share the presentation space. The application is also allowed to copy the presentation space and perform other read-only operations as usual.

WRITE_NONE

The application has exclusive use of the presentation space. No other applications are allowed to share the presentation space, including supervisory applications. The application is allowed to copy the presentation space and perform read-only operations as usual.

READ_WRITE

The application requires only read access to monitor the presentation space and allows other applications that perform read or write, or both, functions to share the presentation space. The application is also allowed to copy the presentation space and perform other read-only operations as usual.

Note: Sharing presentation space is not available between threads in a process.

Table 2. EHLLAPI Read and Write Sharing Option Combinations						
Calling Application	**Super_Write**	**Write_Super**	**Write_Write**	**Write_Read**	**Write_None**	**Read_Write**
Super_Write	Yes	Yes	Yes	No	No	Yes
Write_Super (default)	Yes	No	No	No	No	No
Write_Write	Yes	No	Yes	No	No	Yes
Write_Read	No	No	No	No	No	Yes
Write_None	No	No	No	No	No	No
Read_Write	Yes	No	Yes	Yes	No	Yes

In addition to specifying compatible read and write access options, applications that are designed to work together but cannot allow others to work in the same presentation space can optionally define a keyword, KEY$nnnnnnnn, in the **Set Session Parameters** (9) function call. This keyword allows only those applications that use the same keyword to share the presentation space.

Notes:

1. The **Start Keystroke Intercept** (50) function is nonshareable. Only one application at a time can trap keystrokes.
2. The **Connect To Presentation Space** (1) and **Start Keystroke Intercept** (50) functions share common subsystem functions. Successful requests by an application to share either of these functions can affect the requests of these two functions by other applications. For example, if application A successfully requests a **Connect To Presentation Space** (1) with Write_Read access and KEY$abcdefgh as the keyword, a request by application B to **Connect To Presentation Space** (1) or **Start Keystroke Intercept** (50) is successful only if both applications have set compatible read and write options.

Table 3. Prerequisite Functions and Associated Dependent Functions

Prerequisite Call	Functions	Access
Allocate Communications Buffer (120)	Free Communication Buffer (120)	
Connect Window Service (101)	Change PS Window Name (106) Change Switch List Name (105) Disconnect Window Service (102) Query Window Service (103) Window Status (104)	 Write Read Query=Read Set=Write Write
Connect Presentation Space (1)	Copy Field to String (34) Copy OIA (13) Copy Presentation Space (5) Copy Presentation Space to String (8) Copy String to Field (33) Copy String to Presentation Space (15) Disconnect Presentation Space (2) Find Field Length (32) Find Field Position (31) Query Cursor Location (7) Query Field Attribute (14) Release (12) Reserve (11) Search Field (30) Search Presentation Space (6) Send key (3) Set Cursor (40) Start Playing Macro (110) Wait (4)	Read Read Read Read Write Write Write Read Read Read Read Write Write Read Read Read Write Write Read
Connect Structured Field (120)	Disconnect Structured Field (121) Get Request Completion (125) Read Structured Field (126) Write Structured Field (127)	
Read Structured Field (126)	Get Request Completion (125)	
Start Close Intercept (41)	Query Close Intercept (42) Stop Close Intercept (43)	
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)	
Write Structured Field (127)	Get Request Completion (125)	

Locking Presentation Space

An application, even if specified with shared presentation space, can obtain exclusive control of a presentation space by using the **Lock Presentation Space API** (60) or the **Lock Windows Services API** (61) functions. Requests by the other applications to use a presentation space locked by these functions are queued and processed in first-in-first-out (FIFO) order when the originating application unlocks the presentation space.

If the application that locked the presentation space does not unlock it by using the same call with an **Unlock** option or **Reset System** (21) call, the lock is removed when the application terminates or the session stops.

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 not be capable of producing some codes needed by the another session. ASCII mnemonics that represent these codes can be included in the data string parameter of the **Send Key** (3) function.

The capabilities of the **Send Key** (3) function and the **Get Key** (51) function allow sessions to exchange keystrokes that might not 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 workstation keyboard.

Mnemonics for unshifted keys consist of the escape character followed by an abbreviation. This is also true for the shift keys themselves, Upper 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 of escape character, abbreviation, escape character, abbreviation.

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

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

Upper 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.

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

To represent a key defined as an ASCII character, a 1-byte ASCII value that corresponds 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 PF1, @1 is used. To represent Erase Input, @A@F is used. See the following tables:

@B	Left Tab	@0	Home	@j	PF19
@C	Clear	@1	PF1/F1	@k	PF20
@D	Delete	@2	PF2/F2	@l	PF21
@E	Enter	@3	PF3/F3	@m	PF22
@F	Erase EOF	@4	PF4/F4	@n	PF23
@H	Help (PC400)	@5	PF5/F5	@o	PF24
@I	Insert	@6	PF6/F6	@q	End
@J	Jump	@7	PF7/F7	@s	ScrLk
@L	Cursor Left	@8	PF8/F8	@t	Num Lock
@N	New Line	@9	PF9/F9	@u	Page UP (PC400)
@O	Space	@a	PF10/F10	@v	Page Down (PC400)
@P	Print	@b	PF11/F11	@x	PA1
@R	Reset	@c	PF12/F12	@y	PA2
@T	Right Tab	@d	PF13	@z	PA3
@U	Cursor Up	@e	PF14	@@	@ (at) symbol
@V	Cursor Down	@f	PF15	@$	Alternate Cursor
@X	DBCS	@g	PF16	@<	Backspace
@Y	Caps Lock	@h	PF17		
@Z	Cursor Right	@i	PF18		

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.

@A@C	Test (PC400)	@A@d	Red (PC/3270)
@A@D	Word Delete	@A@e	Pink (PC/3270)
@A@E	Field Exit	@A@f	Green (PC/3270)
@A@F	Erase Input	@A@g	Yellow (PC/3270)
@A@H	System Request	@A@h	Blue (PC/3270)
@A@I	Insert Toggle	@A@i	Turquoise (PC/3270)
@A@J	Cursor Select	@A@j	White (PC/3270)
@A@L	Cursor Left Fast	@A@l	Reset Host Color (PC/3270)
@A@Q	Attention	@A@t	Print (Personal Computer)
@A@R	Device Cancel	@A@y	Forward Word Tab
@A@T	Print Presentation Space	@A@z	Backward Word Tab
@A@U	Cursor Up Fast	@A@-	Field - (PC400)
@A@V	Cursor Down Fast	@A@+	Field + (PC400)
@A@Z	Cursor Right Fast	@A@<	Record Backspace (PC400)
@A@9	Reverse Video	@S@E	Print Presentation Space on Host (PC400)
@A@b	Underscore (PC/3270)	@S@x	Dup
@A@c	Reset Reverse Video (PC/3270)	@S@y	Field Mark

Notes:

1. The first @ symbol in the first table represents the escape character. The first and second @ symbol in the second table is the escape character. The @ symbol is the default escape character. You can change the value of the escape character using the ESC=c option of the **Set Session Parameters** (9) function.

   If you change the escape character to #, the literal sequences used to represent the Backtab, Home, and Erase Input keys become #B, #0, and #A#F, respectively.

   Also, the literal sequence used to represent the @ symbol becomes #@.

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

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

Get Key (51) Function

If the terminal 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. For example, if the **Backtab** key is typed, @B is received. If **PF1** 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.

If the operator types an individual key that is not defined, the **Get Key** (51) function returns a return code of 20 and nothing is sent to the host application.

The **Get Key** (51) function precedes all characters and mnemonics sent to the host application as described in the preceding text with two ASCII characters. The first ASCII character is the PSID of the host presentation space to which the keystrokes are sent. The fourth character is an *A* or *M* for ASCII, special shift, or mnemonic, respectively. See page 81.

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 a return code rejecting the key might result.

Debugging

As an aid in debugging EHLLAPI applications, the Trace Facility of Personal Communications may be used. This facility will produce a log of all EHLLAPI calls, parameters, return values, and return codes. For more information on using the Trace Facility, see the *Personal Communications Quick Beginnings.*

A Simple EHLLAPI Sample Program

The following sample application will enter the character string "Hello World!" in the first input field of host session 'A'.

```
#include  <stdlib.h>
#include  <stdio.h>
#include  <windows.h>
#include  "hapi_c.h"

 int main(char **argv, int argc) {
    int HFunc, HLen, HRc;
   char  HBuff[1];
    struct HLDConnectPS ConnBuff;
    // Send Key string for HOME+string+ENTER:
    char SendString[] = "@0Hello World!@E";

    HFunc = HA_RESET_SYSTEM;
   HLen    =  0;
   HRc     =  0;
    hllapi(&HFunc, HBuff, &HLen, &HRc);
    if (HRc != HARC_SUCCESS) {
        printf("Unable to access EHLLAPI.\n");
       return  1;
   }

    HFunc = HA_CONNECT_PS;
   HLen    =  sizeof(ConnBuff);
   HRc     =  0;
    memset(&ConnBuff, 0x00, sizeof(ConnBuff));
    ConnBuff.stps_shortname = 'A';
    hllapi(&HFunc, (char *)&ConnBuff, &HLen, &HRc);
    switch (HRc) {
      case  HARC_SUCCESS:
      case  HARC_BUSY:
       case HARC_LOCKED: // All these are OK
          break;
      case  HARC_INVALID_PS:
           printf("Host session A does not exist.\n");
          return  1;
      case  HARC_UNAVAILABLE:
           printf("Host session A is in use by another EHLLAPI application.\n");
          return  1;
      case  HARC_SYSTEM_ERROR:
           printf("System error connecting to session A.\n");
          return  1;
      default:
           printf("Error connecting to session A.\n");
          return  1;
   }

    HFunc = HA_SENDKEY;
    HLen    = strlen(SendString)+1; // Length is implicit anyway
   HRc     =  0;
    hllapi(&HFunc, SendString, &HLen, &HRc);
    switch (HRc) {
      case  HARC_SUCCESS:
          break;
      case  HARC_BUSY:
      case  HARC_LOCKED:
           printf("Send failed, host session locked or busy.\n");
          break;
      default:
```

```
            printf("Send  failed.\n");
            break;
        }

        HFunc = HA_DISCONNECT_PS;
        HLen    =  0;
        HRc     =  0;
        hllapi(&HFunc, HBuff, &HLen, &HRc);

        printf("EHLLAPI program ended.\n");
        return  0;
}
```

The following MAKEFILE file could be used to build this application with the IBM Visual Age C/C++ for Windows compiler (assuming the source file is named SAMPLE.C):

```
all: sample.exe

hlldir = C:\PCOMWIN\SAMPLES
hlllib = C:\PCOMWIN\SAMPLES

.SUFFIXES: .C .OBJ

.c.obj:
      icc.exe /Ti /Gh /Gm /Gd /C /I $(hlldir) /Tc $*.c

sample.exe: sample.obj
      ilink.exe /de /noe $(hlllib)\pcscal32.lib $**

sample.obj: sample.c
```

The application could be built with the following command:

```
nmake /a all
```

16/32 Bit Considerations

For platforms on which both 16- and 32-bit EHLLAPI interfaces are supplied you should choose the interface which matches your EHLLAPI application program. If your application is a 16-bit application then it should use the 16-bit interface. If your application is a 32-bit application it should use the 32-bit interface. You choose the interface by linking with either the PCSCALLS.LIB library (16-bit) or the PCSCAL32.LIB library (32-bit).

There is no functional difference between the 16- and 32-bit EHLLAPI interfaces on a given platform. However there are other important differences:

1. The 32-bit EHLLAPI interface extends the presentation space ID (PSID) from 1 byte to 4 bytes. Currently the additional bytes are not used, but your application should set them to binary zeros to insure compatibility with future versions of 32-bit EHLLAPI.
2. The position (offset) of data elements in memory buffers passed to and from EHLLAPI functions are different. Data elements in 32-bit EHLLAPI are aligned to double-word boundaries. Data elements in 16-bit EHLLAPI are not aligned in any particular way. EHLLAPI applications should not be coded to set or retrieve data in the buffers by offset (byte) values. Instead, the supplied data structures in the HAPI_C.H file should be used to set and retrieve data elements. This will

insure that data is set and retrieved from the correct position for both 16- and 32-bit programs.

By prefilling EHLLAPI data buffers with binary zeros, and using the data structures supplied in HAPI_C.H an application can be compiled for 16-bit or 32-bit operation without any source code changes. For example, the following section of code would work for 16-bit EHLLAPI but would fail for 32-bit EHLLAPI:

```
#include  "hapi_c.h"
...
 int Func, Len, Rc;
char  Buff[18];
char  SessType;

 Func = HA_QUERY_SESSION_STATUS;      // Function
 Len    = 18;                                  // Buffer length
Rc     =  0;
 Buff[0] = 'A'                                // Session to query
 hllapi(&Func, Buff, &Len, &Rc);      // Execute function

 SessType = Buff[9];                        // Get session type
...
```

The above example would fail if compiled as a 32-bit EHLLAPI application because:

The application does not set the extended session ID bytes to zero.

The buffer length for this function is 20, not 18.

The session type indicator is not at offset 9 in the data buffer, it is at offset 12.

The following is the same function written to work correctly if compiled for 16 or 32-bit operation. Changed lines are indicated with a ">" mark:

```
   #include  "hapi_c.h"
   ...
    int Func, Len, Rc;
>   struct HLDQuerySessionStatus Buff;
   char  SessType;

    Func = HA_QUERY_SESSION_STATUS;      // Function
>  Len   =  sizeof(Buff);                        //  Buffer  length
   Rc     =  0;
>   memset(&Buff, 0x00, sizeof(Buff));// Zero buffer
>   Buff.qsst_shortname = 'A';               // Session to query
    hllapi(&Func, (char *)&Buff, &Len, &Rc);      // Execute function

>   SessType = Buff.qsst_sestype;          // Get session type
   ...
```

Host Automation Scenarios

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

Host system operation, including:
- Search function
- Sending keystrokes

Distributed processing, including:
- Data extraction
- File transfer

Integrating interfaces

Scenario 1. A 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 programmed operator can use a series of EHLLAPI functions to mimic these actions. After determining the correct starting point for the host system transaction, the programmed operator can call the **Search Presentation Space** (6) function to determine which keyword messages or prompting messages are on the display screen.

Next, the programmed operator can use the **Send Key** (3) function to type data into a host system session and enter a host system transaction. Then the programmed 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 some other 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. Also, while waiting for a response, calling the **Pause** (18) function allows other DOS sessions to share the central 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, your EHLLAPI program could call a customized error message such as:

No Response From Host. Retry?

In this environment, program revisions become very important considerations, because the “programmed 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:

However, because the “programmed 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 that demand attention in designing programs that send keystrokes to the host system. In some application environments, issuing a command is as simple as typing a string and pressing Enter. Other applications involve more complex formatted screens in which data can be entered into any one of several fields. In this environment you must understand the keystrokes required to “fill in” the display screen.

The Tab key mnemonic (@T; see page 18 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, you should be aware of the field lengths and contents. If you fill the fields completely and the next attribute byte is “autoskip,” your cursor will then be moved to the next field. If you then issued a tab, you would skip to yet another field.

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

Scenario 3. Distributed Processing

Some applications fall into the category called “collaborative.” 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 communication 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 workstation 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 4. File Transfer

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 log on to a host system session.

Instead of using one of the 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 5. Automation

An application can provide all the keystrokes for another application or can intersperse keystrokes to the target destination with those from the keyboard. Sometimes, to do this, the application must "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 "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 that are provided to the target application are indistinguishable from the ordinary operator input.

Scenario 6. Keystroke Filtering

An application that acts 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 then be:

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

Figure 1 on page 26 provides a simplified representation of the keystroke flow and the objects within a keyboard enhancement environment.

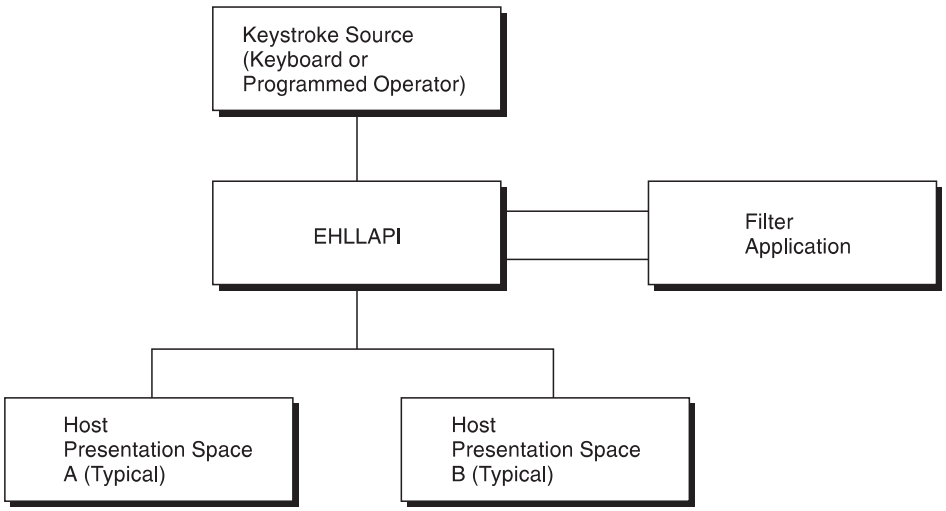


Figure 1. Keystroke Flow

Scenario 7. Keyboard Enhancement

Scenario 8 makes use of filtering to create an **enhancer application program**. An enhancer application program is one that monitors the data coming in from the keyboard and changes it in some specified way. Typically, these application programs use instructions called **keyboard macros**, which 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 single keystroke with a series of keystrokes.

To do this using EHLLAPI, you might 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 three- or four-keystroke command can be condensed into a single keystroke

      Customizing commands so that they are easier to remember or 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 might convert a key combination such as **Alt+Y** into a command to move the cursor to column 35 of the second line in presentation space and write the string “XYZ Tool Corporation, Dallas, Texas”.

c. If a keystroke is rejected, your EHLLAPI program can cause a “beep” to be sounded, using the **Post Intercept Status** (52) function.

4. After your EHLLAPI program exits the filtering loop, **Stop Keystroke Intercept** (53) function to end the filtering process.

Chapter 3. 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. The functions are explained for both the 16-bit and 32-bit environments.

Page Layout Conventions

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

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

Prerequisite Calls

"Prerequisite Calls" lists any calls that must be made prior to calling the function being discussed. If no prerequisite call is required, the word *None* is listed.

Call Parameters

"Call Parameters" 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

"Return Parameters" lists the parameters that must be received by your program after a call to the discussed EHLLAPI function and explains how to interpret those parameters.

Notes on Using This Function

"Notes on Using This Function" lists any session options that affect the function under discussion. It also provides technical information about using the function and application development tips.

Summary of EHLLAPI Functions

The following table is the summary of the EHLLAPI functions:

Number	Function	PC/3270	PC400	VT	Page
1	Connect Presentation Space (1)	Yes	Yes	Yes	38
2	Disconnect Presentation Space (2)	Yes	Yes	Yes	74
3	Send Key (3)	Yes	Yes	Yes	126

© Copyright IBM Corp. 1989, 1997

Number	Function	PC/3270	PC400	VT	Page
4	Wait (4)	Yes	Yes	Yes	157
5	Copy Presentation Space (5)	Yes	Yes	Yes	57
6	Search Presentation Space (6)	Yes	Yes	Yes	121
7	Query Cursor Location (7)	Yes	Yes	Yes	98
8	Copy Presentation Space to String (8)	Yes	Yes	Yes	62
9	Set Session Parameters (9)	Yes	Yes	Yes	135
10	Query Sessions (10)	Yes	Yes	Yes	105
11	Reserve (11)	Yes	Yes	Yes	118
12	Release (12)	Yes	Yes	Yes	117
13	Copy OIA (13)	Yes	Yes	Yes	47
14	Query Field Attribute (14)	Yes	Yes	Yes	99
15	Copy String to Presentation Space (15)	Yes	Yes	Yes	70
18	Pause (18)	Yes	Yes	Yes	89
20	Query System (20)	Yes	Yes	Yes	107
21	Reset System (21)	Yes	Yes	Yes	119
22	Query Session Status (22)	Yes	Yes	Yes	104
23	Start Host Notification (23)	Yes	Yes	Yes	148
24	Query Host Update (24)	Yes	Yes	Yes	101
25	Stop Host Notification (25)	Yes	Yes	Yes	156
30	Search Field (30)	Yes	Yes	Yes	119
31	Find Field Position (31)	Yes	Yes	Yes	77
32	Find Field Length (32)	Yes	Yes	Yes	75
33	Copy String to Field (33)	Yes	Yes	Yes	67
34	Copy Field to String (34)	Yes	Yes	Yes	42
40	Set Cursor (40)	Yes	Yes	Yes	135
41	Start Close Intercept (41)	Yes	Yes	Yes	146
42	Query Close Intercept (42)	Yes	Yes	Yes	95
43	Stop Close Intercept (43)	Yes	Yes	Yes	155
50	Start Keystroke Intercept (50)	Yes	Yes	Yes	151

Number	Function	PC/3270	PC400	VT	Page
51	Get Key (51)	Yes	Yes	Yes	80
52	Post Intercept Status (52)	Yes	Yes	Yes	94
53	Stop Keystroke Intercept (53)	Yes	Yes	Yes	157
60	Lock Presentation Space API (60)	Yes	No	No	85
61	Lock Window Services API (61)	Yes	No	No	87
90	Send File (90)	Yes	Yes	No	123
91	Receive File (91)	Yes	Yes	No	114
99	Convert Position or Convert RowCol (99)	Yes	Yes	Yes	40
101	Connect Window Services (101)	Yes	Yes	Yes	37
102	Disconnect Window Service (102)	Yes	Yes	Yes	73
103	Query Window Coordinates (103)	Yes	Yes	Yes	102
104	Window Status (104)	Yes	Yes	Yes	91
105	Change Switch List LT Name (105)	Yes	Yes	Yes	34
106	Change PS Window Name (106)	Yes	Yes	Yes	32
110	Start Playing Macro (110)	Yes	Yes	Yes	154

Number	Function	PC/3270	PC400	VT	Page
120	Connect for Structured Fields (120)	Yes	No	No	35
121	Disconnect from Structured Fields (121)	Yes	No	No	72
122	Query Communications Buffer Size (122)	Yes	No	No	96
123	Allocate Communications Buffer (123)	Yes	No	No	31
124	Free Communications Buffer (124)	Yes	No	No	79
125	Get Request Completion (125)	Yes	No	No	83
126	Read Structured Fields (126)	Yes	No	No	109
127	Write Structured Fields (127)	Yes	No	No	158

Allocate Communications Buffer (123)

3270	*5250*	*VT*
Yes	No	No

The **Allocate Communications Buffer** function obtains a buffer from Windows memory management. A buffer address must be passed on both the **Read Structured Fields** (126) and **Write Structured Fields** (127) functions.

Prerequisite Calls

None.

Call Parameters

Table 4. Call Parameters

	16-bit	**32-bit**
Function Number	Must be 123	
Data String	See the following table	
Length	Must be 6	Must be 8
PS Position	NA	

The calling data string can contain:

Table 5. Calling Data String

Byte		**Definition**
16-bit	32-bit	
1–2	1–4	32-bit or 16-bit buffer length. (0 < size ≤ (64 KB–256 bytes)=X'FF00')
3–6	5–8	32-bit allocated buffer address (returned)

Return Parameters

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. The EHLLAPI obtains a buffer from Windows 95 and Windows NT memory management and places 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 64 KB minus 256 bytes (X'FF00' bytes) in length.

   See "**Query Communications Buffer Size** (122)" for information regarding buffer size.

2. Buffers obtained using this function must not be shared among different processes. If this is attempted, the applications will experience unpredictable results.
3. An EHLLAPI application must issue a **Free Communications Buffer** (124) function to free the allocated memory.
4. 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.
5. The **Reset System** (21) function frees buffers allocated by this function.

Change PS Window Name (106)

3270	*5250*	*VT*
Yes	Yes	Yes

The **Change PS Window Name** 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

Table 6 (Page 1 of 2). Call Parameters

	16-bit	**32-bit**
Function Number	Must be 106	
Data String	See the following table	

Table 6 (Page 2 of 2). Call Parameters		
	16-bit	**32-bit**
Length	Must be specified (See note.)	Must be 68
PS Position	NA	

Note: The data string length must be specified (normally 3–63 for PC/3270, 4–63 for PC400, 68 for 32-bit).

The calling data string can contain:

Table 7. Calling Data String		
Byte		**Definition**
16-bit	32-bit	
1	1	A 1-character presentation space short name (PSID)
	2–4	Reserved
2	5	A change request option value, select one of: X'01' for changing the presentation space window name. X'02' for resetting the presentation space window name.
3–63	6–66	An ASCII string of from 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.
	67–68	Reserved

Return Parameters

Return Code	Explanation
0	The **Change PS Window Name** function was successful.
1	An incorrect host presentation space short session ID was specified, or the host presentation space 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

A string is ended at the first NULL character found. The NULL character overrides the specified string length. 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. 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.

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)

3270	*5250*	*VT*
Yes	Yes	Yes

The **Change Switch List LT Name** 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 Communications Manager EHLLAPI, and has the same result as the **Change PS Window Name** (106) function.

Prerequisite Calls

Connect Window Services (101).

Call Parameters

Table 8. Call Parameters

	16-bit	**32-bit**
Function Number	Must be 105	
Data String	See the following table	
Length	Normally 4–63	Must be 68
PS Position	NA	

The calling data string can contain:

Table 9. Calling Data String

Byte		**Definition**
16-bit	32-bit	
1	1	A 1-character presentation space short name (PSID)
	2–4	Reserved
2	5	A change request option; select: X'01' for changing a switch list LT name X'02' for resetting a switch list LT name
3–63	6–66	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.
	67–68	Reserved

Return Parameters

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 the host presentation space was not connected.

Return Code	Explanation
2	An error was made in specifying parameters.
9	A system error occurred.
12	The session stopped.

Notes on Using This Function

A string is ended at the first NULL character found. The NULL character overrides the specified string length. 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. 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.

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)

3270	*5250*	*VT*
Yes	No	No

The **Connect for Structured Fields** function allows an application to establish a connection to the emulation program to exchange structured field data with a host application. The workstation 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

Table 10. Call Parameters

	16-bit	**32-bit**
Function Number	Must be 120	
Data String	See the following table	
Length	7 or 11	Must be 16
PS Position	NA	

The calling data string can contain:

Table 11 (Page 1 of 2). Calling Data String

Byte		**Definition**
16-bit	32-bit	

Table 11 (Page 2 of 2). Calling Data String		
Byte		**Definition**
1	1	A 1-character presentation space short name (PSID)
	2–4	Reserved
2–5	5–8	Address of the Query Reply data buffer
6–7	9–10	Destination/origin unique ID. (16-bit word, returned)
	11–12	Reserved
8–11	13–16	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

Return Code	Explanation
0	The **Connect for Structured Fields** function was successful.
1	A specified host presentation space short session ID was not valid, or the host presentation space was not connected.
2	An error was made in specifying parameters.
9	A system error occurred.
10	The function is 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. If the specified DOID is already in use, a return code of 2 will be returned by EHLLAPI.
2. The application should build the Query Reply Data structures in the application's private memory. Refer to Appendix A, 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 Window Services (101)

3270	*5250*	*VT*
Yes	Yes	Yes

The **Connect Window Services** 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

Table 12. Call Parameters

	16-bit	**32-bit**
Function Number	Must be 101	
Data String	1-character short session ID of the host presentation space	
Length	1 is implied	Must be 4
PS Position	NA	

The calling data string can contain:

Table 13. Calling Data String

Byte		**Definition**
16-bit	32-bit	
1	1	A 1-character presentation space short name (PSID)
	2–4	Reserved

Return Parameters

Return Code	**Explanation**
0	The **Connect Window Services** function was successful.

Return Code	Explanation
1	An incorrect host presentation space short session ID was specified, or the Sessions Window Services manager was not connected. This return code could also mean that the API Setting for DDE/EHLLAPI is not set on.
9	A system error occurred.
10	The function is not supported by the emulation program.
11	This resource is unavailable. The host presentation space is already 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.

   A **Connect Window Services** function is sufficient for the process. However, each thread that requests a **Connect Window Services** must have a corresponding **Disconnect Window Services** (102), or one of the threads must issue a **Reset System** (21), which affects all threads and disconnects any remaining connections.

Connect Presentation Space (1)

3270	*5250*	*VT*
Yes	Yes	Yes

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

Prerequisite Calls

None.

Call Parameters

Table 14. Call Parameters

	16-bit	**32-bit**
Function Number	Must be 1	
Data String	1-character short name of the host presentation space	
Length	1 is implied	Must be 4
PS Position	NA	

The calling data string can contain:

Table 15. Calling Data String		
Byte		**Definition**
16-bit	32-bit	
1	1	A 1-character presentation space short name (PSID)
	2–4	Reserved

Return Parameters

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.

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. The specified session either does not exist or is a logical printer session. This return code could also mean that the API Setting for DDE/EHLLAPI is not set on.
4	Successful connection was achieved, but the host presentation space is busy.
5	Successful connection was achieved, but the host presentation space is locked (input inhibited).
9	A system error was encountered.
11	This resource is unavailable. The host presentation space is already being used by another system function.

Notes on Using This Function

1. The **Connect Presentation Space** function is affected by the CONLOG/CONPHYS session option.
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 to issue functions.
3. Each thread that requests a **Connect Presentation Space** must have a corresponding **Disconnect Presentation Space** (2), or one of the threads must issue a **Reset System** (21), which affects all threads and disconnects any remaining connections.
4. More than one EHLLAPI application can share a presentation space, if the applications support sharing (that is, if they were developed to work together and if they exhibit predictable behavior) and have compatible read/write access

and keyword options as set in the **Set Sessions Parameters** (9) function. For more information, see “Set Session Parameters (9)” on page 135.

5. Because the **Connect Presentation Space** and **Start Keystroke Intercept** (50) functions share common subsystem functions, successful requests by an application to share either of these functions for the same session can affect the request of these two functions by other applications. For example, if application A successfully requests a **Connect Presentation Space** for a session with Write_Read access and KEY$abcdefgh as the keyword, a request by application B to **Connect Presentation Space** for a session and **Start Keystroke Intercept** is successful only if both applications have set compatible read/write options.
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. Refer to *Personal Communications Version 4.2 Quick Beginnings for Windows 95 and Windows NT.*

Convert Position or Convert RowCol (99)

3270	*5250*	*VT*
Yes	Yes	Yes

The **Convert Position** or **Convert RowCol** function converts the host presentation space positional value into the display row and column coordinates or converts the display row and column coordinates into the host presentation space positional value. This function does not change the cursor position.

Prerequisite Calls

None.

Call Parameters

Table 16 (Page 1 of 2). Call Parameters

	16-bit	**32-bit**
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 is specified as the second character in the data string parameter. The 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 42. NA when P is specified as the second character in the data string parameter.	

Table 16 (Page 2 of 2). Call Parameters		
	16-bit	**32-bit**
PS Position	Column, when R is specified as the second character in the data string parameter. The lower limit for valid input is 1. The upper limit for valid input ranges from 24 to 43 depending on how your host presentation space is configured. See "Notes on Using This Function" on page 42. Host presentation space position, when P is specified as the second character in the data string parameter. The lower limit for valid input is 1. The upper limit for valid input ranges from 1920 to 3564 depending on how your host presentation space is configured. See "Notes on Using This Function" on page 42.	

The calling data string can contain:

Table 17. Calling Data String		
Byte		**Definition**
16-bit	32-bit	
1	1	A 1-character presentation space short name (PSID)
	2–4	Reserved
2	5	**Convert** option P or R
	6–8	Reserved

Return Parameters

This function returns a length and a return code.

Length:

For the **Convert Position** function (P as the second character in the 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 the second character in the 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 the second character of the 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 the second character of the 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 defined:

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 *Personal Communications Version 4.2 Quick Beginnings for Windows 95 and Windows NT.*
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 104.

Copy Field to String (34)

3270	*5250*	*VT*
Yes	Yes	Yes

The **Copy Field to String** 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

Table 18. Call Parameters

	16-bit	32-bit
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. **DBCS Only:** When Extended Attributes Double-byte (EAD) option is specified, the length of the data string must be at least three times the length of the field. When both EAB and EAD options are specified, the length of the data string must be at least four times the length of the field.	
Length	Number of bytes to copy (the length of the data string).	
PS Position	Identifies the target field. This can 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 presentation space. The first byte in the returned data string is the beginning byte of the identified field in the host presentation space. 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 presentation space

Return Code: The following codes are defined:

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 not valid.

Return Code	Explanation
9	A system error was encountered.
24	Unformatted host presentation space.

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 when one of the following conditions is encountered:

   When the end of the field is reached
   When the length of the target string is exceeded
3. **DBCS Only:** If the target string is ended at the higher byte of the DBCS character, the byte is translated into a blank. If the EAD option is set to on, three bytes are returned for each character. If both the EAB and EAD options are set to on, four bytes are returned for each character.

   Note: When the field wraps at the end of the presentation space, wrapping occurs when the end of the presentation space is reached.
4. **DBCS Only:** The **Set Session Parameters** (9) function EAD option is used with this function to return a 2-byte EAD. If the EAD option is specified instead of the EAB option, EAD is returned preceding each character. If both the EAB and EAD options are specified, EAD is returned preceding the EAB.
5. 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.
6. The **Copy Field to String** function is affected by the ATTRB/NOATTRB/NULLATTRB, the ATTRB/NOATTRB/NULLATTRB (NULLATTRB is valid for Windows 95 and Windows NT only), the EAB/NOEAB, the XLATE/NOXLATE, the DISPLAY/NODISPLAY, the DISPLAY/NODISPLAY (for Windows 95 and Windows NT only), the EAD/NOEAD (for DBCS only), and the NOSO/SPACESO/SO (for DBCS only) session options. Refer to items 5 on page 138; 13 and 14 on page 141; 17 on page 142; and 20 and 21 on page 143 for more information.

   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 and NOEAD	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.

EAD — Double-byte character set attributes are returned as shown in the following tables.

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 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
5–6	Character attributes 00 = Default value 11 = Double byte character
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. The following table 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
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)

Double-byte character set attributes (for DBCS only)

– The first byte

Bit Position	Character Position	Field Attribute Position
0	Double-byte character	Reserved
1	The first byte of the double-byte character	Reserved
2	SO	Reserved
3–4	SI (Bit position 3)	5250 DBCS related field When the value of bit position 7 is 0: 00 = Default 01 = DBCS only 10 = Either DBCS or SBCS 11 = Mixture of DBCS and SBCS When the value of bit position 7 is 1: 00 = Reserved 01 = DBCS only without SO/SI 10 = Reserved 11 = Reserved
5	Reserved	SO/SI enable (3270 only)

Bit Position	Character Position	Field Attribute Position
6	Reserved	Character attributes exist (3270 only)
7	Reserved	5250 DBCS related extended field 0 = Basic double-byte field 1 = Extended double-byte field

– The second byte

Bit Position	Character Position	Field Attribute Position
0	Reserved	Left grid line (3270 only)
1	Reserved	Upper grid line (3270 only)
2	Reserved	Right grid line (3270 only)
3	Reserved	Under grid line (3270 only)
4	Left grid line	Left grid line
5	Upper grid line	Upper grid line
6–7	Reserved	Reserved

For a PS/2 monochrome display, the characters in the application (workstation) 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.

7. 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 Allocation” on page 7 for more information.

Note: 5250 emulation supports a presentation space of 24 rows by 80 columns. In some instances, Communications Manager 5250 emulation displays a 25th row. This occurs when either an error message from the host is displayed or when the operator selects the SysReq key. Personal Communications for OS/2 always displays the same information on the 24th row. By **EXTEND_PS** option, an EHLLAPI application can use the same interface with Communications Manager EHLLAPI and valid presentation space is extended when this condition occurs.

Copy OIA (13)

3270	*5250*	*VT*
Yes	Yes	Yes

The **Copy OIA** 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 workstation and the host.

Prerequisite Calls

Connect Presentation Space (1).

Call Parameters

Table 19. Call Parameters

	16-bit	32-bit
Function Number	Must be 13	
Data String	Preallocated target data string	
Length	103	104
PS Position	NA	

Return Parameters

This function returns a data string and a return code.

Data String: A 103-byte string for 16–bit and 104–byte string for 32–bit. See "Format of the Returned OIA Data String" on page 49 for more information.

Return Code: The following codes are defined:

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 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 Allocation" on page 7 for more information.

Format of the Returned OIA Data String

The OIA data string contains the following information:

Table 20. Calling Data String		
Byte		**Definition**
16-bit	32-bit	
1	1	The OIA format byte. The value is 1 (PC/3270), 9 (PC400), or 5 (VT).
2–81	2–81	The OIA image in the host code points.
82–103	82–103	OIA group indicator meanings.
	104	Reserved.

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 on page 50 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 4.2 Quick Beginnings for Windows 95 and Windows NT* 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 through 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 on page 50, find the OIA graphics character corresponding to each hexadecimal value found in step 2.

Note: Group 8 (byte 0) machine, communications, and program check images are followed by a three-digit number related to the type of check.

The online and screen ownership group images are for non-SNA 3274 controller configurations. For SNA, the CD hex value is translated by CD (see Figure 2 on page 50). If running on a 3174 controller or SDLC connection, the hex value X'F4' is replaced by X'B2' or X'22'. The highlight indicator is a corresponding image (in the first 80 bytes of the data string) of the “Group 5 (offset 86: Highlight group 1” byte. The highlight indicator is followed by either X'F9' (blink), X'FC' (underscore), X'D2' (reverse video), or X'80' (host default).

The short session ID followed by X'20' is in column 7.

All group images are represented by Main Frame Interactive (MFI) hex code points.

Note: The OIA image data string position minus 1 position equals the OIA column.

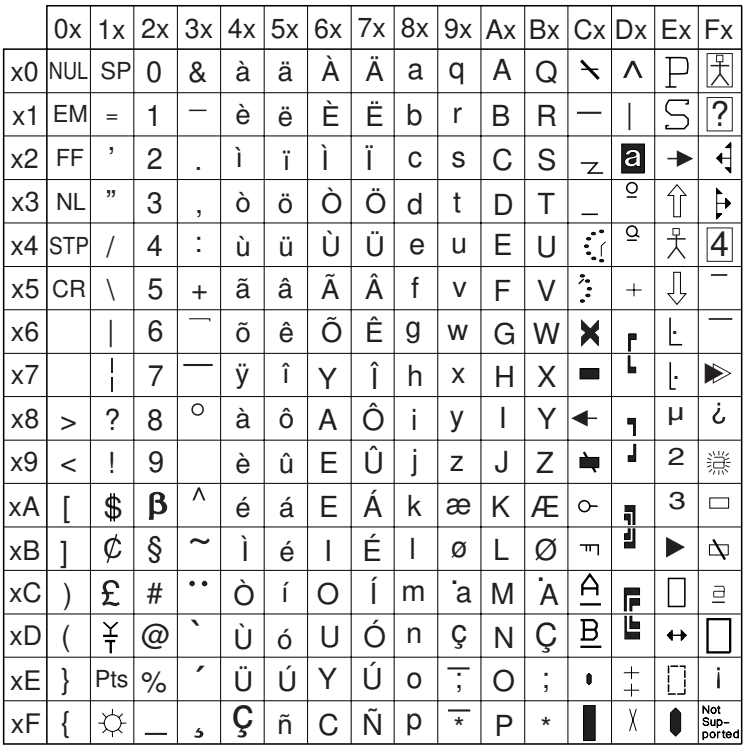


Figure 2. Host Presentation Space Characters

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	Katakana (Japan only)
3	Alphanumeric
4–5	Reserved
6	Hiragana (Japan only)
7	Double-byte character

Group 3 (Offset 84): Shift State

Bit	Meaning
0	Upper shift

Bit	Meaning
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	Terminal wait
2	Minus symbol

Bit	Meaning
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	

VT Host OIA Group Indicator Meanings and Its Image

Details of the VT Host OIA group are listed in the following tables.

Group 1 (Offset 82): Online and Screen Ownership

Bit	Meaning
5	Subsystem ready

Group 2 (Offset 83): Character Selection

Bit	Meaning
0	Upper shift
2	CAPS

Group 7 (Offset 88): Insert

Bit	Meaning
0	Insert mode

Some columns on the OIA line display different messages for VT than those messages displayed for 3270/5250. See the following table for specific details.

Column	Symbol
1–7	VT220 7
	VT220 8
	VT100
	VT52
	VTANSI
9 - 12	LOCK
61 - 64	HOLD

Copy Presentation Space (5)

3270	*5250*	*VT*
Yes	Yes	Yes

The **Copy Presentation Space** 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, you can override this translation with the ATTRB option under the **Set Session Parameters** (9) function.

Prerequisite Calls

Connect Presentation Space (1).

Call Parameters

Table 21. Call Parameters

	16-bit	**32-bit**
Function Number	Must be 5	
Data String	Preallocated target string the size of your host presentation space. This 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 length of the presentation space. **DBCS Only:** When the EAD option is specified, the length of the data string must be at least three times the length of the presentation space. When both the EAB and EAD options are specified, the length of the data string must be at least four times the length of the presention space.	
Length	NA (the length of the host presentation space 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 defined:

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. **DBCS Only:** The **Set Session Parameters** (9) function EAD option is used with this function to return a 2-byte EAD. If the EAD option is specified instead of the EAB option, EAD is returned preceding each character. If both the EAB and EAD options are specified, EAD is returned preceding the EAB.

   If the start position of the copy is at the second byte in the double-byte character, or the end position is at the first byte in the double-byte character, the bytes are translated into blanks.
3. The **Copy Presentation Space** function is affected by the following session options:

   ATTRB/NOATTRB/NULLATTRB,
   EAB/NOEAB,
   XLATE/NOXLATE,
   BLANK/NOBLANK,
   DISPLAY/NODISPLAY,
   EAD/NOEAD
   NOSO/SPACESO/SO (for DBCS only)

Refer to items 5 on page 138; 13, 14, 15 and 17 on page 142; and 20 and 21 on page 143 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) 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 and NOEAD	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.
EAD	Double-byte character set attributes are returned as shown in the following tables.
NOSO/SPACESO/SO	When NOSO is specified, it works as SPACESO. The size of the presentation space is not changed.

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 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
5–6	Character attribute 00 = Default value 11 = Double-byte character
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. The following table 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
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)

Double-byte character set attributes (for DBCS only)

– The first byte

Bit Position	Character Position	Field Attribute Position
0	Double-byte character	Reserved
1	The first byte of the double-byte character	Reserved
2	SO	Reserved
3–4	SI (Bit position 3)	5250 DBCS related field When the value of bit position 7 is 0: 00 = Default 01 = DBCS only 10 = Either DBCS or SBCS 11 = Mixture of DBCS and SBCS When the value of bit position 7 is 1: 00 = Reserved 01 = DBCS only without SO/SI 10 = Reserved 11 = Reserved
5	Reserved	SO/SI enabled (3270 only)
6	Reserved	Character attributes exist (3270 only)
7	Reserved	5250 DBCS related extended field 0 = Basic double-byte field 1 = Extended double-byte field

– The second byte

Bit Position	Character Position	Field Attribute Position
0	Reserved	Left grid line (3270 only)
1	Reserved	Upper grid line (3270 only)
2	Reserved	Right grid line (3270 only)
3	Reserved	Under grid line (3270 only)
4	Left grid line	Left grid line
5	Upper grid line	Upper grid line
6–7	Reserved	Reserved

For a PS/2 monochrome display, the characters in the application (workstation) 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.

If you want to copy only a portion of the host presentation space, use the **Copy Presentation Space to String** (8) 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 Allocation” on page 7 for more information.

Note: 5250 emulation supports a presentation space of 24 rows by 80 columns. In some instances, Communications Manager 5250 emulation displays a 25th row. This occurs when either an error message from the host is displayed or when the operator selects the SysReq key. Personal Communications for Windows 95 always displays the same information on the 24th row. By the **EXTEND_PS** option, an EHLLAPI application can use the same interface with Communications Manager EHLLAPI and valid presentation space is extended when this condition occurs.

Copy Presentation Space to String (8)

3270	*5250*	*VT*
Yes	Yes	Yes

The **Copy Presentation Space to String** 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

Table 22 (Page 1 of 2). Call Parameters

	16-bit	**32-bit**
Function Number	Must be 8	
Data String	Preallocated target string the size of your host presentation space. 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 presentation space. **DBCS Only:** When the EAD option is specified, the length of the data string must be at least three times the length of the presentation space. When both the EAB and EAD options are specified, the length of the data string must be at least four times the length of the presention space.	
Length	Length of the target data string.	

Table 22 (Page 2 of 2). Call Parameters		
	16-bit	**32-bit**
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 presentation space.
Return Code: The following codes are defined:

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 presentation space.
4	The host presentation space contents were copied. The host presentation space was waiting for host response.
5	The host presentation space was copied. The keyboard was locked.
7	The host presentation space position is not valid.
9	A system error was encountered.

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. **DBCS Only:** The **Set Session Parameters** (9) function EAD option is used with this function to return a 2-byte EAD. If the EAD option is specified instead of the EAB option, EAD is returned preceding each character. If both the EAB and EAD options are specified, EAD is returned preceding the EAB.

   If the start position of the copy is at the second byte in the double-byte character, or the end position is at the first byte in the double-byte character, the bytes are translated into blanks. If the EAD option is set to on, three bytes are returned for each character. If both the EAB and EAD options are set to on, four bytes are returned for each character.
3. The **Copy Presentation Space to String** function is affected by the following options:

   ATTRB/NOATTRB/NULLATTRB
   ATTRB/NOATTRB/NULLATTRB

EAB/NOEAB
XLATE/NOXLATE
BLANK/NOBLANK
DISPLAY/NODISPLAY
EAD/NOEAD (for DBCS only)
NOSO/SPACESO/SO (for DBCS only)

Refer to items 5 on page 138; 13 and 14 on page 141; 15 on page 141; 17 on page 142; and 20 and 21 on page 143

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 and NOEAD	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.
EAD	Double-byte character set attributes are returned as shown in the following tables.

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 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

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

Bit Position	Meaning
0-3	Reserved
4	Bold 1 = On 0 = Off
5	Underscore 1 = On 1 = Off
6	Blink 1 = On 0 = Off
7	Reverse 0 = On 1 = Off

The following table shows PC/3270 and PC400 character color attributes. The following table 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
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)

Double-byte character set attributes

– The first byte

Bit Position	Character Position	Field Attribute Position
0	Double-byte character	Reserved
1	The first byte of the double-byte character	Reserved
2	SO	Reserved
3–4	SI (Bit position 3)	5250 DBCS related field When the value of bit position 7 is 0: 00 = Default 01 = DBCS only 10 = Either DBCS or SBCS 11 = Mixture of DBCS and SBCS When the value of bit position 7 is 1: 00 = Reserved 01 = DBCS only without SO/SI 10 = Reserved 11 = Reserved
5	Reserved	SO/SI enable (3270 only)
6	Reserved	Character Attributes exist (3270 only)

Bit Position	Character Position	Field Attribute Position
7	Reserved	5250 DBCS related extended field 0 = Basic double-byte field 1 = Extended double-byte field

– The second byte

Bit Position	Character Position	Field Attribute Position
0	Reserved	Left grid line (3270 only)
1	Reserved	Upper grid line (3270 only)
2	Reserved	Right grid line (3270 only)
3	Reserved	Under grid line (3270 only)
4	Left grid line	Left grid line
5	Upper grid line	Upper grid line
6–7	Reserved	Reserved

For a PS/2 monochrome display, the characters in the application (workstation) 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, 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 Allocation” on page 7 for more information.

Note: 5250 emulation supports a presentation space of 24 rows by 80 columns. In some instances, Communications Manager 5250 emulation displays a 25th row. This occurs when either an error message from the host is displayed or when the operator selects the SysReq key. Personal Communications for Windows 95 always displays the same information on the 24th row. By the **EXTEND_PS** option, an EHLLAPI application can use the same interface with Communications Manager EHLLAPI and valid presentation space is extended when this condition occurs.

Copy String to Field (33)

3270	*5250*	*VT*
Yes	Yes	Yes

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

Prerequisite Calls

Connect Presentation Space (1).

Call Parameters

Table 23. Call Parameters

	16-bit	32-bit
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. This can be the PS position of any byte within the target field. Copy always starts at the beginning of the field.	

Return Parameters

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 incorrect data was sent to the target field (such as a field attribute).
6	Copy was completed, but data is truncated.
7	The host presentation space position is not valid.
9	A system error was encountered.
24	Unformatted host presentation space.

Notes on Using This Function

1. The **Copy String to Field** function is affected by the following options:

   STRLEN/STREOT
   EOT
   EAB/NOEAB
   XLATE/NOXLATE
   PUTEAB/NOPUTEAB

   Refer to items 1 and 2 on page 137; 13 and 14 on page 141; 18 on page 142; and 20 and 21 on page 143 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 135).

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 presentation space 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.
5. **DBCS Only:** Double-byte characters can be included as a part of the string.

   Note: PC400 does not add SO and SI to the string. When you write the strings, including double-byte characters at the DBCS mixed field, generate SO and SI and create the area where double-byte characters are written by using the **Send Key** (3) function in advance.

   If both single-byte and double-byte characters exist in a string, the data might be truncated because the data length in EBCDIC is longer than in JISCII. In this case, only the first byte or the second byte of the double-byte character is not written.

   If the last character in the original string is the first byte of the double-byte character, the character is not written and not counted in the length.

   A control character is converted from single-byte character to double-byte character, or from double-byte character to single-byte character depending on the field condition. A pair of NULL+Control Character between SO and SI is treated as a double-byte control character. For example, the following strings are copied into the single-byte character field or the double-byte character field:

String	Meanings	Single-byte character field	Double-byte character field
X'000C'	(NULL)(FF) X'00'X'0C'	(SB NULL)(SB FF) X'00'X'0C'	(DB NULL)(DB FF) X'0000'X'000C'
X'0E000C0F'	(SO)(DB FF)(SI) X'0E'X'000C'X'0F'	–S error	(DB FF) X'000C'
Note: SB means single-byte characters and DB means double-byte characters.			

Note: 5250 emulation supports a presentation space of 24 rows by 80 columns. In some instances, Communications Manager 5250 emulation displays a 25th row. This occurs when either an error message from the host is displayed or when the operator selects the SysReq key. Personal Communications for Windows 95 always displays the same information on the 24th row. By the **EXTEND_PS** option, an EHLLAPI application can use the same interface with Communications Manager EHLLAPI and valid presentation space is extended when this condition occurs.

Copy String to Presentation Space (15)

3270	*5250*	*VT*
Yes	Yes	Yes

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

Prerequisite Calls

Connect Presentation Space (1).

Call Parameters

Table 24. Call Parameters

	16-bit	**32-bit**
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

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 incorrect data was sent to the 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 not valid.
9	A system error was encountered.

Notes on Using This Function

1. The **Copy String to Presentation Space** function is affected by the following options:

   STRLEN/STREOT
   EOT
   EAB/NOEAB
   XLATE/NOXLATE
   PUTEAB/NOPUTEAB the EAD/NOEAD (for DBCS only), and the NOSO/SPACESO/SO (for DBCS only)

   Refer to items 1 and 2 on page 137; 13 and 14 on page 141; 18 on page 142; and 20 and 21 on page 143 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 135).
4. Although the **Send Key** (3) function accomplishes the same purpose, this function responds 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 with 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.
7. **DBCS Only:** Double-byte characters can be included as a part of the string.

   Note: PC400 does not add SO and SI to the string. When you write the strings, including double-byte characters at the DBCS mixed field, generate SO and SI and create the area where double-byte characters are written by using the **Send Key** (3) function in advance.

   If both single-byte and double-byte characters exist in a string, the data might be truncated because the data length in EBCDIC is longer than in JISCII. If only the first byte or the second byte of the double-byte character must be written into the string, a blank is written.

   If the last character in the original string is the first byte of the double-byte character, the character is not written and not counted in the length.

   If the character to be written into the last character of the target presentation space is SO/SI or the first byte of the double-byte character, the character is not written and truncated, and not counted in the length.

   A control character is converted from single-byte character to double-byte character, or from double-byte character to single-byte character depending on the field condition. A pair of NULL+Control Character between SO and SI is treated as a double-byte control character. For example, the following strings are copied into the single-byte character field or the double-byte character field:

String	Meanings	Single-byte character field	Double-byte character field
X'000C'	(NULL)(FF) X'00'X'0C'	(SB NULL)(SB FF) X'00'X'0C'	(DB NULL)(DB FF) X'0000'X'000C'
X'0E000C0F'	(SO)(DB FF)(SI) X'0E'X'000C'X'0F'	–S error	(DB FF) X'000C'
Note: SB means single-byte characters and DB means double-byte characters.			

Note: 5250 emulation supports a presentation space of 24 rows by 80 columns. In some instances, Communications Manager 5250 emulation displays a 25th row. This occurs when either an error message from the host is displayed or when the operator selects the SysReq key. Personal Communications for Windows 95 always displays the same information on the 24th row. By the **EXTEND_PS** option, an EHLLAPI application can use the same interface with Communications Manager EHLLAPI and valid presentation space is extended when this condition occurs.

8. This function call may cause a cursor movement to an unexpected position with some host applications. A SendKey function may be a better choice for filling a field than this function.

   Note: This only occurs with VT sessions or connections to an ASCII host.

Disconnect from Structured Fields (121)

3270	*5250*	*VT*
Yes	No	No

The **Disconnect from Structured Fields** 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

Table 25. Call Parameters

	16-bit	32-bit
Function Number	Must be 121	
Data String	See the following table	
Length	Must be 3	Must be 8
PS Position	NA	

Data String Contents

Table 26. Calling Data String

Byte		Definition
16-bit	32-bit	
1	1	A 1-character presentation space short name (PSID).
	2–4	Reserved.
2–3	5–6	Destination/origin unique ID returned by the Connect for structured field (120) functions.
	7–8	Reserved.

Return Parameters

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. When a **Disconnect from Structured Fields** function is called, any outstanding asynchronous **Read Structured Fields** (126) or **Write Structured Fields** (127) function requests are returned if the application issues the **Get Request Completion** (125) function call. Use the asynchronous form of this function when cleaning up after issuing a Disconnect call.
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 Window Service (102)

3270	*5250*	*VT*
Yes	Yes	Yes

The **Disconnect Window Service** 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

Table 27. Call Parameters

	16-bit	32-bit
Function Number	Must be 102	
Data String	See the following table	
Length	1	4
PS Position	NA	

Data String Contents

Table 28. Calling Data String

Byte		Definition
16-bit	32-bit	
1	1	A 1-character presentation space short name (PSID)
	2–4	Reserved

Return Parameters

Return Code	Explanation
0	The **Disconnect Window Service** function was successful.
1	Your program is not connected for Window Services.
9	A system error occurred.

Notes on Using This Function

After the **Disconnect Window Service** function has been called, your application no longer manages the presentation space window.

Before exiting the application, you should 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.

Disconnect Presentation Space (2)

3270	*5250*	*VT*
Yes	Yes	Yes

The **Disconnect Presentation Space** 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

Table 29. Call Parameters		
	16-bit	**32-bit**
Function Number	Must be 2	
Data String	NA	
Length	NA	
PS Position	NA	

Return Parameters

Return Code	Explanation
0	The **Disconnect Presentation Space** function was successful.
1	Your program was not currently 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, the **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.

Find Field Length (32)

3270	*5250*	*VT*
Yes	Yes	Yes

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

Table 30. Call Parameters

	16-bit	32-bit
Function Number	Must be 32	
Data String	See the following table	
Length	NA	NA
PS Position	See note below	

Note: **PS Position:** Identifies the field within the host presentation space at which to start the **Find**. It 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
bb or T*b*	This field
P*b*	The previous field, either protected or unprotected.
N*b*	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 *b* symbol represents a required blank.

Return Parameters

This function returns a length and a return code.

Length: The following 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 the host presentation space.

Return Code: The following codes are defined:

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 not valid.
9	A system error was encountered.
24	No such field was found.
28	Field length of 0 bytes.

Notes on Using This Function

Except when bb or Tb 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)

3270	*5250*	*VT*
Yes	Yes	Yes

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

Prerequisite Calls

Connect Presentation Space (1).

Call Parameters

Table 31. Call Parameters

	16-bit	**32-bit**
Function Number	Must be 31	
Data String	See the following table	
Length	NA	NA
PS Position	See note below	

Note: **PS Position:** Identifies the field within the host presentation space at which to start the **Find**. It can be the PS position of any byte within the field in which you want the **Find** to start.

The calling 2-character data string can contain:

Code	Explanation
bb or T*b*	This field

Code	Explanation
P*b*	The previous field, either protected or unprotected
N*b*	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 *b* symbol represents a required blank.

Return Parameters

This function returns a length and a return code.

Length: The following 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 the requested field from the origin of the host presentation space. This position is defined to be the first position after the attribute byte.

Return Code: The following codes are defined:

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 not valid.
9	A system error was encountered.
24	No such field was found.
28	Field length of 0 bytes.

Notes on Using This Function

Except when bb or Tb 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)

3270	*5250*	*VT*
Yes	No	No

The **Free Communications Buffer** function returns to Windows 95 and Windows NT 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

Table 32. Call Parameters

	16-bit	**32-bit**
Function Number	Must be 124	
Data String	See the following table	
Length	Must be 6	Must be 8
PS Position	NA	

Data String Contents

Table 33. Calling Data String

Byte		**Definition**
16-bit	32-bit	
1–2	1–4	Must be 0
3–6	5–8	The address of the buffer

Return Parameters

Return Code	**Explanation**
0	The **Free Communications Buffer** function was successful.
2	An error was made in specifying parameters.
9	A system error occurred.
41	The buffer is in use.

Notes on Using This Function

1. If the application attempts to free an “in use” buffer, the free request will be denied and a return code of 41 will be 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)

3270	*5250*	*VT*
Yes	Yes	Yes

The **Get Key** 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

Table 34. Call Parameters

	16-bit	**32-bit**
Function Number	Must be 51	
Data String	See the following table	
Length	8	12
PS Position	NA	

Data String Contents

Table 35. Calling Data String

Byte		**Definition**
16-bit	32-bit	
1	1	One of the following values: A 1-character presentation space short name (PSID) A blank or null indicating a function call for the host-connected presentation
	2–4	Reserved
2–8	5–11	Blanks that hold space for the symbolic representation of the requested data
	12	Reserved

Return Parameters

This function returns a data string and a return code.

Data String: See the following table:

Table 36 (Page 1 of 2). Calling Data String

Byte		**Definition**
16-bit	32-bit	

Table 36 (Page 2 of 2). Calling Data String		
Byte		**Definition**
1	1	One of the following values: A 1-character presentation space short name (PSID). A blank or null indicating a function call for the host-connected presentation.
	2–4	Reserved.
2	5	An option code character, one of the following characters: A for ASCII returned. M for keystroke mnemonic.
3–8	6–11	These 6 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'. Bytes 3 and 4 contains 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 17). Bytes 5 through 8 might 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').

For clarification, some examples of returned data strings are provided below:

Note: 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.

16-Bit Interface

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').

32-Bit Interface

EbbbAt E is the presentation space short name. The keystrokes are returned as ASCII (A), and the returned key is the lowercase letter t. (Bytes 7–11 = X'00').

EbbbM@2 E is the presentation space short name. The keystrokes are returned as mnemonics, and the returned key is PF2 (Bytes 8–11 = X'00').

Return Code: The following 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, either:

   Increase the value of the calling length parameter for the **Start Keystroke Intercept** (50) function, or

   Execute 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, FIFO), the three bytes that it occupied are made 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. 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).
6. 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 Allocation” on page 7 for more information.

Get Request Completion (125)

3270	*5250*	*VT*
Yes	No	No

The **Get Request Completion** 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”) completion on a previous function call such as **Read Structured Fields** (126) or **Write Structured Fields** (127).

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 for 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 for 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 for 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

Table 37. Call Parameters

	16-bit	**32-bit**
Function Number	Must be 125	
Data String	See the following table	
Length	Must be 14	Must be 24
PS Position	NA	

Data String Contents

Table 38 (Page 1 of 2). Calling Data String

Byte		**Definition**
16-bit	32-bit	
1	1	A 1-character presentation space short name (PSID)
	2–4	Reserved

Table 38 (Page 2 of 2). Calling Data String		
Byte		**Definition**
2	5	N or W N=NOWAIT is required W=WAIT is required
	6–8	Reserved
3–4	9–10	Function request ID.
5–6	11–12	Reserved
7–10	13–16	Reserved
11–12	17–20	Reserved
13–14	21–24	Reserved

The **Get Request Completion** function behaves differently depending upon the second character of the parameter string, which is one of the following characters:

N Nowait option: If a specific request ID was supplied and the function has completed, control will be 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 has completed, control will be 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 has not completed, the call will wait until the function has completed before returning to the application. If the supplied request ID was zero and no eligible asynchronous function has completed, the call will wait until a function completes before returning to the calling application. On return, the return code value will be zero and the data string will be completed as defined below.

Return Parameters

Table 39. Calling Data String		
Byte		**Definition**
16-bit	32-bit	
5–6	11–12	Function number of the completed asynchronous function (126 or 127). (returned)
7–10	13–16	Address of the data string of the completed asynchronous function call. (The application must not reuse the data string until the request has completed). (returned)
11–12	17–20	Length of the data string of the completed asynchronous function call. (returned)
13–14	21–24	Return code of the completed asynchronous function call. (returned)

Return Code	Explanation
0	The **Get Request Completion** function was successful.
2	An error was made in specifying parameters.

Return Code	Explanation
9	A system error was encountered.
38	Requested function was not complete.
42	No matching request was found.

There are some differences between return codes 38 and 42:

1. Return code 38
   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
   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" for "Asynchronous") on a previous function call such as **Read Structured Fields** or **Write Structured Fields**.
2. If the return code is a 0, the application should check the returned data string for information pertaining to the completion of the requested asynchronous function.

Lock Presentation Space API (60)

3270	*5250*	*VT*
Yes	No	No

The **Lock Presentation Space API** function allows the application to obtain or release exclusive control of the presentation space window over other Windows 95 and Windows NT applications. While locked, no other application can connect to the presentation space window.

Successful processing of this function with the Lock causes EHLLAPI presentation space window functions requested from other EHLLAPI applications to be queued until the requesting application unlocks the presentation space. Requests from the locking application are processed normally.

Prerequisite Calls

Connect to Presentation Space (1)

Call Parameters

Table 40. Call Parameters

	16-bit	**32-bit**
Function Number	Must be 60	
Data String	See the following table	
Length	Must be 3	Must be 8
PS Position	NA	

Data String Contents

Table 41. Calling Data String

Byte		**Definition**
16-bit	32-bit	
1	1	A 1-character presentation space short name (PSID).
	2–4	Reserved.
2	5	One of the following characters: L to lock the API. U to unlock the API.
3	6	One of the following characters: R to return if the presentation space is already locked by an application. Q to queue the Lock request if the presentation space is already locked by an application.
	7–8	Reserved.

Return Parameters

Return Code	**Explanation**
0	The **Lock Presentation Space API** 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 was encountered.
43	The API was already locked by another EHLLAPI application (on LOCK) or API not locked (on UNLOCK).

Notes on Using This Function

The following EHLLAPI functions are queued when a lock is in effect:

Send Key (3)
Copy Presentation Space (5)
Search Presentation Space (6)
Copy Presentation Space to String (8)
Release (11)
Reserve (12)
Query Field Attribute (14)
Copy String to Presentation Space (15)
Search Field (30)
Find Field Position (31)
Find Field Length (32)
Copy String to Field (33)
Copy Field to String (34)
Set Cursor (40)
Send File (90)
Receive File (91)
Connect to Presentation Space (1) with the CONPHYS parameter set in a previous **Set Sessions Parameter** (9) function call.

These queued requests are not serviced until the lock is removed. When the lock is removed, the queued requests are processed in first-in-first-out (FIFO) order. EHLLAPI functions not listed are run as if there was no lock. The requesting application unlocks the presentation space window by one of the following methods:

Disconnecting from the presentation space while still owning the Lock.
Issuing the **Reset System** (21) function while still owning the Lock.
Stopping the application while still owning the Lock.
Stopping the session.
Successfully issuing the **Lock Presentation Space API** with the Unlock option.

Before exiting the application, you should unlock any presentation space windows that have been locked with the **Lock Presentation Space API** function. If the application exits with outstanding locks, or a **Reset System** (21), or **Disconnect Presentation Space** (2) function is issued, the locks are released.

It is recommended that applications lock the presentation space only for short periods of time and only when exclusive use of the presentation space is required.

Lock Window Services API (61)

3270	*5250*	*VT*
Yes	No	No

The **Lock Window Services API** function allows the application to obtain or release exclusive control of the presentation space window over other Windows 95 and Windows NT applications. While locked, no other application can connect to the presentation space window.

Successful processing of this function with the Lock causes EHLLAPI presentation space window functions requested from other EHLLAPI applications to be queued

until the requesting application unlocks the presentation space. Requests from the locking application are processed normally.

Prerequisite Calls

Connect Window Services (101)

Call Parameters

Table 42. Call Parameters

	16-bit	32-bit
Function Number	Must be 61	
Data String	See the following table	
Length	Must be 3	Must be 8
PS Position	NA	

Data String Contents

Table 43. Calling Data String

Byte		Definition
16-bit	32-bit	
1	1	A 1-character presentation space short name (PSID).
	2–4	Reserved.
2	5	One of the following characters: L to lock the API. U to unlock the API.
3	6	One of the following characters: R to return if the presentation space is already locked by an application. Q to queue the Lock request if the presentation space is already locked by an application.
5–6	11–12	Function number of the completed asynchronous function (126 or 127). (returned)
	7–8	Reserved.

Return Parameters

Return Code	Explanation
0	The **Lock Window Services API** 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 was encountered.
38	Requested function was not complete.
43	The API was already locked by another EHLLAPI application (on LOCK) or API not locked (on UNLOCK).

Notes on Using This Function

The following EHLLAPI functions are queued when a lock is in effect:

Window Status (104)
Change Switch List Name (105)
Change PS Window Name (106)

These queued requests are not serviced until the lock is removed. When the lock is removed, the queued requests are processed in first-in-first-out (FIFO) order.

The requesting application unlocks the presentation space window by one of the following methods:

Successfully issuing the **Lock Window Services API** with the UNLOCK option.
Disconnecting from the presentation space while still owning the Lock.
Issuing the **Reset System** (21) function while still owning the Lock.
Stopping the application while still owning the Lock.
Stopping the session.

Before exiting the application, you should Unlock any presentation space windows that have been locked with the **Lock Window Services API**function. If the application exits with outstanding locks, the subsystem releases the locks.

It is recommended that applications lock the presentation space only for short periods of time and only when exclusive use of the presentation space is required.

Pause (18)

3270	*5250*	*VT*
Yes	Yes	Yes

The **Pause** function waits for a specified amount of time. It should be used in place of *timing loops* to wait for an event to occur. A **Pause** 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

None .

Call Parameters

Table 44. Call Parameters

	16-bit	**32-bit**
Function Number	Must be 18	
Data String	NA	
Length	Contains the pause duration in half-second increments	
PS Position	NA	

Return Parameters

Return Code	Definition
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 to get 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 138 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 0028 (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 completed.
4. A practical maximum value for the **Pause** function is 2400. You should 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 because the time interval created by a **Pause** function is approximate.

   Set the time interval to zero in a loop.
5. IPAUSE set and the interruptible pause allow an EHLLAPI application to determine whether the specified host presentation space (PS) or operator information area (OIA) is updated. The following 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, you can make an application wait until the host presentation space 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 occurred.

Window Status (104)

3270	*5250*	*VT*
Yes	Yes	Yes

The **Window Status** function allows the application to query or change a window's presentation space size, location, or visible state.

Prerequisite Calls

Connect Window Services (101).

Call Parameters

Table 45. Call Parameters

	16-bit	**32-bit**
Function Number	Must be 104	
Data String	See the following table	
Length	16 or 20	24 or 28
PS Position	NA	

The calling data string can contain:

Table 46. Calling Data String

Byte		**Definition**
16-bit	32-bit	
1	1	A 1-character presentation space short name (PSID)
	2–4	Reserved
2	5	A request option value, select one of the following values: X'01' for set status **Note:** When the session is embedded In-Place in a compound OLE document, the set form of this function (byte 5 = X'01') always returns 0 but has no effect. X'02' for query for status X'03' for query for extended status
	6	Reserved

If the request option value is X'01' (set status):

Table 47 (Page 1 of 2). Calling Data String

Byte		**Definition**
16-bit	32-bit	

Table 47 (Page 2 of 2). Calling Data String		
Byte		**Definition**
3–4	7–8	A 16- or 32-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 the 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.)
5–6	9–12	A 16- or 32-bit word containing the X window position coordinate. (Ignored if the move option is not set.)
7–8	13–16	A 16- or 32-bit word containing the Y window position coordinate. (Ignored if the move option is not set.)
9–10	17–20	A 16- or 32-bit word containing the X window size in device units. (Ignored if the size option is not set.)
11–12	21–24	A 16- or 32-bit word containing the Y window size in device units. (Ignored if the size option is not set.)
13–16	25–28	A 16- or 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 as follows: 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):

Table 48 (Page 1 of 2). Calling Data String		
Byte		**Definition**
16-bit	32-bit	

Table 48 (Page 2 of 2). Calling Data String		
Byte		**Definition**
3–4	7–8	A 16- or 32-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.
5–6	9–12	A 16- or 32-bit word containing the X window position coordinate. (Ignored if the move option is not set.)
7–8	13–16	A 16- or 32-bit word containing the Y window position coordinate. (Ignored if the move option is not set.)
9–10	17–20	A 16- or 32-bit word containing the X window size in device units. (Ignored if the size option is not set.)
11–12	21–24	A 16- or 32-bit word containing the Y window size in device units. (Ignored if the size option is not set.)
13–16	25–28	A 16- or 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 as follows: 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) the return status will be the handle of the emulator presentation space window:

Table 49 (Page 1 of 2). Calling Data String		
Byte		**Definition**
16-bit	32-bit	
3–4	7–8	A 16- or 32-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.
5–6	9–10	A 16- or 32-bit word containing the current font size in the X-dimension. The value is in screen pels.
7–8	11–12	A 16- or 32-bit word containing the current font size in the Y-dimension. The value is in screen pels.
9–12	13–16	Reserved. This value is always zero.

Table 49 (Page 2 of 2). Calling Data String		
Byte		**Definition**
13–14	17–18	A 16- or 32-bit word containing the row number of the first visible character of the presentation space. This value is usually 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.
15–16	19–20	A 16- or 32-bit word containing the column number of the first visible character of the presentation space.
17–20	21–24	A 16- or 32-bit word containing the presentation space window handle of the session.

Return Parameters

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 Presentation Manager window position and size. All x and y coordinate positions and sizes are in pels.

Post Intercept Status (52)

3270	*5250*	*VT*
Yes	Yes	Yes

The **Post Intercept Status** 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.

Prerequisite Calls

Start Keystroke Intercept (50).

Call Parameters

Table 50. Call Parameters		
	16-bit	**32-bit**
Function Number	Must be 52	
Data String	See the following table	
Length	Must be 2	Must be 8
PS Position	NA	

The calling data string can contain:

Table 51. Calling Data String		
Byte		**Definition**
16-bit	32-bit	
1	1	One of the following values: The 1-letter short name of the presentation space. A blank or null indicating a function call for the host-connected presentation space.
	2–4	Reserved
2	5	One of the following characters: A for accepted keystroke. R for rejected keystroke.
	6–8	Reserved.

Return Parameters

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)

3270	*5250*	*VT*
Yes	Yes	Yes

The **Query Close Intercept** function allows the application to determine if the close option was selected.

Prerequisite Calls

Start Close Intercept (41).

Call Parameters

Table 52. Call Parameters

	16-bit	32-bit
Function Number	Must be 42	
Data String	1-character short session ID of the host presentation space	
Length	Must be 1	Must be 4
PS Position	NA	

The calling data string can contain:

Table 53. Calling Data String

Byte		Definition
16-bit	32-bit	
1	1	A 1-character presentation space short name (PSID)
	2–4	Reserved

Return Parameters

Return Code	Explanation
0	A close intercept event did not occur.
1	The presentation source was not valid.
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)

3270	*5250*	*VT*
Yes	No	No

The **Query Communications Buffer Size** function allows an application to determine both the maximum and the optimum buffer sizes supported by the emulation program.

Prerequisite Calls

None.

Call Parameters

Table 54. Call Parameters		
	16-bit	**32-bit**
Function Number	Must be 122	
Data String	See the following table	
Length	Must be 9	Must be 20
PS Position	NA	

The calling data string can contain:

Table 55. Calling Data String		
Byte		**Definition**
16-bit	32-bit	
1	1	A 1-character presentation space short name (PSID)
	2–4	Reserved
2–3	5–8	16- or 32-bit field for the optimum supported inbound buffer size (Returned value)
4–5	9–12	16- or 32-bit field for the maximum supported inbound buffer size (Returned value)
6–7	13–16	16- or 32-bit field for the optimum supported outbound buffer size (Returned value)
8–9	17–20	16- or 32-bit field for the maximum supported outbound buffer size (Returned value)

Return Parameters

Return Code	Explanation
0	The **Query Communications Buffer Size** 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.
10	The function was not supported by the emulation program.

Notes on Using This Function

1. There is no way to require the user to use this function. It is not a required function so that 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 Structured Fields** 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) with the buffer sizes returned by the **Query Communications Buffer Size** minus 1 byte. For destination/origin connections, the 8-byte header supplied in the **Read** and **Write Structured Fields** 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) with 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 workstation hardware and by the emulator. The maximum buffer size can be used only 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 workstation 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)

3270	*5250*	*VT*
Yes	Yes	Yes

The **Query Cursor Location** 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

Table 56. Call Parameters

	16-bit	**32-bit**
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 defined:

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)

3270	*5250*	*VT*
Yes	Yes	Yes

The **Query Field Attribute** function returns the attribute byte of the field containing the input host presentation space 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

Table 57. Call Parameters

	16-bit	**32-bit**
Function Number	Must be 14.	
Data String	NA.	
Length	NA.	
PS Position	Identifies the target. This 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 defined:

Return Code	Explanation
0	The **Query Field Attribute** was successful.
1	Your program is not currently connected to a host session.
7	The host presentation space position is not valid.
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
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 = Nonfield attribute flag 1 = Field attribute flag
1	Visibility 0 = Nondisplay 1 = Display

Bit Position	Meaning
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)

3270	*5250*	*VT*
Yes	Yes	Yes

The **Query Host Update** function lets the programmed operator determine if the host has updated the host presentation space or OIA because:

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

Table 58. Call Parameters

	16-bit	**32-bit**
Function Number	Must be 24	
Data String	1-character short name of the host presentation space, or a blank or null indicating request for updates to host-connected presentation space	
Length	1 is implied	4 is implied
PS Position	NA	

The calling data string can contain:

Table 59. Calling Data String		
Byte		**Definition**
16-bit	32-bit	
1	1	A 1-character presentation space short name (PSID)
	2–4	Reserved

Return Parameters

Return Code	Definition
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.

Query Window Coordinates (103)

3270	*5250*	*VT*
Yes	Yes	Yes

The **Query Window Coordinates** 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 window.

Prerequisite Calls

Connect Window Services (101).

Call Parameters

Table 60 (Page 1 of 2). Call Parameters		
	16-bit	**32-bit**
Function Number	Must be 103	

Table 60 (Page 2 of 2). Call Parameters		
	16-bit	**32-bit**
Data String	1-character short session ID of the host presentation space	
Length	17 is implied	20 is implied
PS Position	NA	

The calling data string can contain:

Table 61. Calling Data String		
Byte		**Definition**
16-bit	32bit	
1	1	One of the following values: A 1-character presentation space short name (PSID) A blank or null indicating a function call for the current connection presentation space
	2–4	Reserved
2-17	5–20	Reserved

Return Parameters

This function returns a data string and a return code.

Table 62. Calling Data String		
Byte		**Definition**
16-bit	32-bit	
1	1	One of the following values: A 1-character presentation space short session ID A blank or null indicating a function call for the current connection presentation space
	2–4	Reserved
2–17	5–20	Four 32-bit unsigned integers that return:
2–5	5–8	XLeft Long integer in pels of the left X coordinate of the rectangular window relative to the desktop window
6–9	9–12	YBottom Long integer in pels of the bottom Y coordinate of the rectangular window relative to the desktop window
10–13	13–15	XRight Long integer in pels of the right X coordinate of the rectangular window relative to the desktop window
14–17	16–20	YTop Long integer in pels of the top Y coordinate of the rectangular window relative to the desktop window

Return Code: The following codes are defined:

Return Code	Explanation
0	The **Query Window Coordinates** function was successful.
1	Your program was not currently connected to the host session.
9	A system error occurred.
12	The session stopped.

Query Session Status (22)

3270	*5250*	*VT*
Yes	Yes	Yes

The **Query Session Status** function is used to obtain session-specific information.

Prerequisite Calls

None.

Call Parameters

Table 63. Call Parameters

	16-bit	**32-bit**
Function Number	Must be 22.	
Data String	An 18/20-byte string consisting of a 1-byte short name of the target presentation space plus 17 bytes for returned data. Position 1 can be filled with: 1. A blank or a null to indicate a request for the host_connected presentation space. 2. An * (asterisk) to indicate a request for the keyboard-owner presentation space.	
Length	Must be 18.	Must be 20.
PS Position	NA.	

Return Parameters

This function returns a data string and a return code.

Table 64 (Page 1 of 2). Calling Data String

Byte		**Definition**
16-bit	32-bit	
1	1	A 1-character presentation space short name (PSID)
	2–4	Reserved
2–9	5–12	Session long name (same as short name)

Table 64 (Page 2 of 2). Calling Data String		
Byte		**Definition**
10	13	**Session** **Type** **D** 3270 display **E** 3270 printer **F** 5250 display **G** 5250 printer **H** ASCII VT
11	14	Session characteristics expressed by a binary number including the following session-characteristics bits 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
12–13	15–16	Number of rows in the host presentation space, expressed as a binary number
14–15	17–18	Number of columns in the host presentation space, expressed as a binary number
16–17	19–20	Host code page expressed as a binary number
18		Reserved

Return Code: The following codes are defined:

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. See “Memory Allocation” on page 7 for more information.

Query Sessions (10)

3270	*5250*	*VT*
Yes	Yes	Yes

The **Query Sessions** function returns a 16-byte (12-byte for 16-bit) data string describing each host session.

Prerequisite Calls

None.

Call Parameters

Table 65. Call Parameters		
Function	**Description**	
	16-bit	32-bit
Function Number	Must be 10	
Data String	Preallocated string of 16*n*bytes long (12*n*for 16-bit) (*n* =number of sessions) or more	
Length	12*n*bytes	16*n*bytes
PS Position	NA	

Note: When the length is not matched to the number of sessions, the return code is 2.

Return Parameters

This function returns a data string, a length, and a return code.

Data String: The returned data string is 16*n* bytes long (12*n* for 16-bit), where *n* is the number of host sessions. The descriptors are concatenated into the data string and each session type, and presentation space size of a host session.

The format of each 16-byte (12-byte for 16-bit) session descriptor is as follows:

Table 66. Calling Data String		
Byte		**Definition**
16-bit	32-bit	
1	1	A 1-character presentation space short name (PSID)
	2–4	Reserved
2–9	5–12	Session long name (same as short name)
10	13	Connection type H=host
	14	Reserved
11–12	15–16	Host presentation space size (this is a binary number and is not in display format). If the session type is a print session, the value is 0.

Length: The number of host sessions started.

Return Code: The following codes are defined:

Return Code	Explanation
0	The **Query Sessions** function was successful.
2	An incorrect string length was made.

Return Code	Explanation
9	A system error was encountered.

Notes on Using This Function

1. If an application program receives RC=2 or RC=0, the number of the active sessions is returned in the length field. The application program can recognize the minimum string length by this number.
2. The **Query Sessions** function is affected by the CFGSIZE/NOCFGZISE session option. See item 16 on page 142 for more information.

Notes:

1. When NOCFGSIZE is set in **Set Session Parameters**(9) for a 5250 session, the value of presentation space size returned in byte position 11 and 12 from **Query Sessions**(10) will be changed in accordance with the selection of EXTEND_PS or NOEXTEND_PS.
2. When EXTEND_PS is set in **Set Session Parameters**(9), presentation space size returned from **Query Sessions**(10) will include the size of the message line, if it exists.
3. When NOEXTEND_PS is set, the value will not change regardless of the existence of a message line. In the case of 25 row, 80 column presentation space, the value can be 1920 or 2000.

Query System (20)

3270	*5250*	*VT*
Yes	Yes	Yes

The **Query System** function can be used by an EHLLAPI application program to determine the level of PC/3270 or PC400 support 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 below under “Return Parameters.”

Prerequisite Calls

None.

Call Parameters

Table 67. Call Parameters

	16-bit	32-bit
Function Number	Must be 20	
Data String	Preallocated string of 35 bytes	36 bytes
Length	Must be 35	Must be 36
PS Position	NA	

Return Parameters

This function returns a data string and a return code.

Data String: A data string of 35 bytes (for 16–bit) or 36 bytes (for 32–bit) is returned. The bytes are defined as follows:

Table 68. Calling Data String

Byte		Definition
16-bit	32-bit	
1	1	EHLLAPI version number
2–3	2–3	EHLLAPI level number
4–9	4–9	Reserved
10–12	10–12	Reserved
13	13	Hardware base, U=Unable to determine
14	14	Program type, where P=IBM Personal Communications
15–16	15–16	Reserved
17–18	17–18	PCOMM version/level as a 2-byte ASCII value
19	19	Reserved
20–23	20–23	Reserved
24–27	24–27	Reserved
28–29	28–29	Reserved
	30	Reserved
30–31	31–32	NLS type expressed as a 2-byte binary number
32	33	1-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=XGA M=Monochrome V=VGA U=Unknown
33–35	34–36	Reserved

Return Code

The following codes are defined:

Return Code	Explanation
0	The **Query System** function was successful; data string has been returned.
1	EHLLAPI is not loaded. (PC/3270 only)

Return Code	Explanation
2	An incorrect string length was specified. (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. See“Memory Allocation” on page 7 for more information.

Read Structured Fields (126)

3270	*5250*	*VT*
Yes	No	No

The **Read Structured Fields** 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 the **Read Structured Fields** is completed. If the call specifies “A” (for Asynchronous), the application receives control immediately after the call. If the call specifies “M” (for Asynchronous, message mode), the application receives control immediately after the call. The application can wait for the message. In any case (“S”, “A”, or “M”), the application provides the buffer address in which the data from the host is to be placed.

For a successful asynchronous completion of this function, the following statements 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, 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 returns the following information:

1. Function request ID
2. Address of the data string from the asynchronous request
3. Length of the data string
4. Return code of the completed function

Prerequisite Calls

Connect for Structured Fields (120) and **Allocate Communication Buffer** (123).

Call Parameters

Table 69 (Page 1 of 2). Call Parameters

	16-bit	**32-bit**
Function Number	Must be 126	
Data String	See the following table	
Length	8, 10 or 14	20

Table 69 (Page 2 of 2). Call Parameters		
	16-bit	**32-bit**
PS Postion	NA	

The calling data string can contain:

Table 70. Calling Data String		
Byte		**Definition**
32-bit	16-bit	
1	1	A 1-character presentation space short name (PSID).
	2–4	Reserved.
2	5	S or 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, can wait for the event object. M = Asynchronous. Control is returned immediately to the application, can wait for the message.
	6	Reserved.
3–4	7–8	2-byte destination/origin ID.
5–8	9–12	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.
9–10	13–16	Reserved.
11–12	17–20	When “M” is specified in position 2 the window handle of the window that receives the message should be set. The message is a return value of RegisterWindowMessage (“PCSHLL”)(not equal 0).
13–14		The data in these positions is ignored by EHLLAPI. However, no erro 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 5, (2 for 16-bit) and the function is completed successfully, the following data string is returned:

Table 71 (Page 1 of 2). Call Parameters		
Byte		**Definition**
16-bit	32-bit	

Table 71 (Page 2 of 2). Call Parameters		
Byte		**Definition**
9–10	13–14	2-byte function request ID. It is used by the **Get Request Completion** (125) function to determine the completion of this function call.
	15–16	Reserved.
	17–20	4-byte value in which the event object address is returned by EHLLAPI. The application can wait for this event object. When the event object is cleared, the application must issue the **Get Request Completion** (125) function call (32-bit only).

Note: A event object address is returned for each successful asynchronous request. The event object should not be used again. A new event object is returned for each request and is valid for only the duration of that request.

Data String: If “M” (asynchronous message mode) is specified in position 5 (2 for 16-bit applications) and the function is completed successfully, the following data string is returned:

Table 72. Return Parameters		
Byte		**Definition**
9–10	13–14	A 2-byte function request ID. It is used by the **Get Request Completion (125)** function to determine the completion of this function call.
	15–16	Reserved.
11–12	17–18	Task ID of asynchronous message mode.
	19–20	Reserved.

Note: If the function is completed successfully, an application window receive 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.

Return Code: The following codes are defined:

Return Code	Explanation
0	The **Read 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.
11	Resource unavailable (memory unavailable).

Return Code	Explanation
35	Request rejected. An outbound transmission from the host was canceled.
36	Request rejected. Lost contact with the host.
37	The function was successful, but the host is inbound disabled.

Notes on Using This Function

1. Return code 35 will be 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 communication with the host. Corrective action is the responsibility of the application.
3. Return code 37 will be 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 (RC=11) is returned if more than 20 asynchronous requests are attempted.
5. If you are using an IBM Global Network connection, the maximum number of asynchronous requests is 10.

The structured field data contains the application structured 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 of data in the message, the number does not include the buffer header prefix, which contains 8 bytes). This value is returned by EHLLAPI.
4	1 word	n (buffer size: the supplied length of the data buffer that does include the 8-byte message header). This value must be set by the application.
6	1 word	X'C000'.
8	8 bytes	Length of the first (or only) structured field message.
10	1 byte	First nonlength byte of the structured field message.
		⋮
m+7	1 byte	Last byte in the structured field message.

Bytes 0 through 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 through *m*+7 are used for the structured field message received from the host (which could 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.

Synchronous Requests

When **Read Structured Fields** is requested synchronously (the S option in the data string), control is returned to the application only after the request is satisfied. The application can assume:

The return code is correct.
The data in the communications buffer (read buffer) is correct.
The host is no longer processing the **Read Structured Fields** request.

Asynchronous Requests

When **Read Structured Fields** is requested asynchronously (the A option in the data string), the application *cannot* assume:

The return code is correct.
The data in the communications buffer (read buffer) is correct.
The host is no longer processing the **Read Structured Fields** request.

When requested asynchronously, EHLLAPI returns the following values:

A 16-bit Request ID in positions 13–14 (9–10 for 16-bit) of the data string
The address of a event object in positions 17—20 of the data string

These are used to complete the asynchronous **Read Structured Fields** call.

The following steps must be completed to determine the outcome of an asynchronous **Read Structured Fields** function call:

If the EHLLAPI return code is not zero, the request failed. No asynchronous request has been made. The application must take appropriate actions before attempting the call again.

If the return code is zero, the application should wait until the event object is in the signaled state by using the **Get Request Completion** (125) function or **Wait For SingleObject**. The event object should not be reused. The event object is valid only for the duration of the **Read Structured Fields** function call through the completion of the **Get Request Completion** (125) function call.

Once the event object is in the signaled state, use the returned 16-bit Request ID as the Request ID parameter in a call to the **Get Request Completion** (125) function. The data string returned from the **Get Request Completion** (125) function call contains the final return code of the **Read Structured Fields** function call.

When **Read Structured Fields** is requested asynchronously (the M option in the data string), the application *cannot* assume:

The return code is correct.
The data in the communications buffer (read buffer) is correct.
The host is no longer processing the **Read Structured Fields** request.

When requested asynchronously with the M option, EHLLAPI returns the following values:

A 16-bit Request ID in positions 13–14 (9–10 for 16-bit) of the data string
Task ID of asynchronous message mode in positions 17–18 (11–12 for 16-bit) of the data string.

These are used to complete the asynchronous **Read Structured Fields** call.

Receive File (91)

3270	*5250*	*VT*
Yes	Yes	No

The **Receive File** function is used to transfer a file from the host session to the workstation 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

Table 73. Call Parameters

	16-bit	32-bit
Function Number	Must be 91.	
Data String	Refer to the examples.	
Length	Length, in number of bytes, of the data string. Overridden if in EOT mode.	

Following are examples of the data strings for SBSC:

3270 Session

To receive the file from the VM/CMS host system:

pc_filename [*id:*]*fn ft* [*fm*] [(*option*]

To receive the file from the MVS/TSO host system:

pc_filename[*id:*]*dataset* [(*member*)] [*/password*] [*option*]

To receive the file from the CICS host system:

pc_filename [*id:*]*host_filename* [(*option*]

5250 Session

To receive the file from the AS/400 host system:

pc_filename [*id:*]*library file member* [*option*]

Following are examples of the data strings for DBCS:

3270 Session

To receive the file from the VM/CMS host system:

pc_filename [*id:*]*fn ft* [*fm*] [(*option*]

To receive the file from the MVS/TSO host system:

pc_filename [*id:*]*dataset* [(*member*)] [*/password*]
[(*option*]

To receive the file from the CICS host system:

pc_filename [*id:*]*host_filename* [(*option*]

5250 Session

To receive the file from the AS/400 host system:

pc_filename [*id:*]*library file member* [*option*]

Note: Parameters within [ ] are optional. Available options are listed below. For more information on the options, refer to *Personal Communications Version 4.2 Reference for Windows 95 and Windows NT.*

Host System	Common Options
VM/CMS	ASCII, JISCII, CRLF, APPEND, TIME n, CLEAR, NOCLEAR, PROGRESS, QUIET
MVS/TSO	ASCII, JISCII, CRLF, APPEND, TIME (n), CLEAR, NOCLEAR, PROGRESS, QUIET, AVBLOCK|TRACKS|CYLINDERS
CICS	ASCII, JISCII, CRLF, NOCRLF, BINARY, TIME n, CLEAR, NOCLEAR, PROGRESS, QUIET
OS/400	ASCII, JISCII, CRLF, APPEND, TIME n, CLEAR, NOCLEAR, PROGRESS, QUIET

Note: JISCII is valid in a DBCS session for Japan only and ASCII is valid for all other SBCS and DBCS sessions.

Other options specified will be passed to the host transfer program. The file transfer program on the host side either uses them, ignores them, or returns an error. Consult the host transfer program documentation to see a complete list of the options supported.

Return Parameters

Return Code	Explanation
2	Parameter error or you have specified a length that is too long (more than 255 bytes) 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 timeout set by the **Set Session Parameter** (9) function.
101	File transfer was successful (transfer to/from CICS).

If you receive return code 2 or 9, there is a problem with the system or with the way you specified your data string.

Other return codes can also be received, which relate to message numbers generated by the host transfer program. For transfers to a CICS host transfer program, subtract 100 from the return code to give you the numeric portion of the message. For example, a return code of 101 would mean that the message number INW0001 was issued by the host. For other host transfer programs, just use the return code as the numerical part of the message. For example, a return of 34 would mean that message TRANS34 was issued by the host transfer program. The documentation for your host transfer program should give more information about the meanings of the specific messages.

Win32 error codes reported by EHLLAPI are greater than 300. To determine the Win32 error code, subtract 300 and refer to the *Win32 Programmer's 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=0 session options. See items 1 and 2 on page 137 and items 7 and 8 on page 139 for more information.
2. 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.
3. 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 *Personal Communications Version 4.2 Quick Beginnings for Windows 95 and Windows NT.*

Release (12)

3270	*5250*	*VT*
Yes	Yes	Yes

The **Release** 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

Table 74. Call Parameters

	16-bit	**32-bit**
Function Number	Must be 12	
Data String	NA	
Length	NA	
PS Postion	NA	

Return Parameters

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)

3270	*5250*	*VT*
Yes	Yes	Yes

The **Reserve** function locks the keyboard that is associated with the host-connected presentation space to block input from the terminal operator.

The reserved host presentation space remains locked until one of the following occurs:

Connect (1) function is executed to a new session.
Disconnect Presentation Space (2) function is executed.
Release (12) function is executed.
Reset System (21) function is executed.
Start Keystroke Intercept (50) function is executed.
EHLLAPI application program is terminated.

Prerequisite Calls

Connect Presentation Space (1).

Call Parameters

Table 75. Call Parameters

	16-bit	**32-bit**
Function Number	Must be 11	
Data String	NA	
Length	NA	
PS Postion	NA	

Return Parameters

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 gaining access to that session until your application processing is complete.
2. The keyboard input that a user makes while the keyboard is locked by this function is enqueued and processed after the session is terminated.
3. This function locks both the mouse and the keyboard input. The application program must unlock the presentation space to enable either the mouse or the keyboard input.

Reset System (21)

3270	*5250*	*VT*
Yes	Yes	Yes

The **Reset System** 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.

You can 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

Table 76. Call Parameters

	16-bit	**32-bit**
Function Number	Must be 21	
Data String	NA	
Length	NA	
PS Postion	NA	

Return Parameters

Return Code	**Definition**
0	The **Reset System** function was successful.
1	EHLLAPI is not loaded.
9	A system error was encountered.

Notes on Using this Function

For the 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)

3270	*5250*	*VT*
Yes	Yes	Yes

The **Search Field** 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* and 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 presentation space is reached.

Prerequisite Calls

Connect Presentation Space (1).

Call Parameters

Table 77. Call Parameters

	16-bit	32-bit
Function Number	Must be 30.	
Data String	Target string for search.	
Length	Length of the target data string. Overridden in EOT mode.	
PS Postion	Identifies the target field. For SRCHALL, this can be the PS position of any byte within the target field. For SRCHFROM, it is the beginning point of the search for SRCHFRWD or the ending point of the search for SRCHBKWD. See note 3 on page 121.	

Return Parameters

This function returns a length and a return code.

Length: The following codes are defined:

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 defined:

Return Code	Explanation
0	The **Search Field** function was successful.
1	Your program is not connected to a host session.
2	Parameter error. Either the string length was zero, or EOT mode was specified but no EOT character was found in calling data string.
7	The host presentation space position is not valid.
9	A system error was encountered.

Return Code	Explanation
24	The search string was not found, or the host presentation space 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 on page 137 through 4 on page 138 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 the 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. **DBCS Only:** 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 for.

   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').

Note: 5250 emulation supports a presentation space of 24 rows by 80 columns. In some instances, Communications Manager 5250 emulation displays a 25th row. This occurs when either an error message from the host is displayed or when the operator selects the SysReq key. Personal Communications for Windows 95 and Windows NT always displays the same information on the 24th row. By **EXTEND_PS** option, an EHLLAPI application can use the same interface with Communications Manager EHLLAPI and valid presentation space is extended when this condition occurs.

Search Presentation Space (6)

3270	*5250*	*VT*
Yes	Yes	Yes

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

Prerequisite Calls

Connect Presentation Space (1).

Call Parameters

Table 78. Call Parameters

	16-bit	32-bit
Function Number	Must be 6.	
Data String	Target string for search.	
Length	Length of the target data string. Overridden in EOT mode.	
PS Postion	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 defined:

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 defined:

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 not valid.
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 on page 137 through 4 on page 138 through 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 also 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. **DBCS Only:** 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 for.

   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').

Note: 5250 emulation supports a presentation space of 24 rows by 80 columns. In some instances, Communications Manager 5250 emulation displays a 25th row. This occurs when either an error message from the host is displayed or when the operator selects the SysReq key. Personal Communications for Windows 95 and Windows NT always displays the same information on the 24th row. By **EXTEND_PS** option, an EHLLAPI application can use the same interface with Communications Manager EHLLAPI and valid presentation space is extended when this condition occurs.

Send File (90)

3270	*5250*	*VT*
Yes	Yes	No

The **Send File** function is used to transfer a file from the workstation session where EHLLAPI is running to a host session.

Prerequisite Calls

None.

Call Parameters

Table 79. Call Parameters

	16-bit	32-bit
Function Number	Must be 90.	
Data String	Refer to the examples.	
Length	Length of the target data string. Overridden in EOT mode.	
PS Postion	Must be 0.	

Following are examples of the data strings for SBCS

3270 Session

To send the file to the VM/CMS host system:

pc_filename [*id:*]*fn ft* [*fm*] [(*option*]

To send the file to the MVS/TSO host system:

pc_filename [*id:*]*dataset* [(*member*)] [*/password*] [*option*]

To send the file to the CICS host system:

pc_filename [*id:*]*host_filename* [(*option*]

5250 Session

To send the file to the AS/400 host system:

pc_filename [*id:*]*library file member* [*option*]

Following are examples of the data strings for DBCS:

3270 Session

To send the file to the VM/CMS host system:

pc_filename [*id:*]*fn ft* [*fm*] [(*option*]

To send the file to the MVS/TSO host system:

pc_filename [*id:*]*dataset* [(*member*)] [*/password*] [(*option*]

To send the file to the CICS host system:

pc_filename [*id:*]*host_filename* [(*option*]

5250 Session

To send the file to the AS/400 host system:

pc_filename [*id:*]*library file member* [*option*]

Note: Parameters within [ ] are optional. Available options are listed below. For more information of the options, refer to *Personal Communications Version 4.2 Reference for Windows 95 and Windows NT.*

Host System	Common Options
VM/CMS	ASCII, JISCII, CRLF, APPEND, LRECL n, RECFM v|f, TIME n, CLEAR, NOCLEAR, PROGRESS, QUIET
MVS/TSO	ASCII, JISCII, CRLF, APPEND, LRECL (n), RECFM (v|f|u), TIME (n), CLEAR, NOCLEAR, PROGRESS, QUIET, BLKSIZE (n), SPACE (n[,m]), AVBLOCK|TRACKS|CYLINDERS
CICS	ASCII, JISCII, CRLF, BINARY, TIME n, CLEAR, NOCLEAR, PROGRESS, QUIET
OS/400	ASCII, JISCII, CRLF, APPEND, SRC, LRECL n, TIME n, CLEAR, NOCLEAR, PROGRESS, QUIET

Note:

JISCII is valid in a DBCS session for Japan only and ASCII is valid for all other SBCS and DBCS sessions.

Other options specified will be passed to the host transfer program. The file transfer program on the host side either uses them, ignores them, or returns an error. Consult the host transfer program documentation to see a complete list of the options supported.

Return Parameters

Return Code	Explanation
2	Parameter error or you have specified a length that is too long (more than 255 bytes) 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 not valid 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 timeout set by the **Set Session Parameter** (9) function.
101	File transfer was successful (transfer to/from CICS).

If you receive return code 2 or 9, there is a problem with the system or with the way you specified your data string.

Other return codes can also be received which relate to message numbers generated by the host transfer program. For transfers to a CICS host transfer program, subtract 100 from the return code to give you the numeric portion of the message. For example, a return code of 101 would mean that the message number INW0001 was issued by the host. For other host transfer programs, just use the return code as the numerical part of the message. For example, a return of 34 would mean that message TRANS34 was issued by the host transfer program. The documentation for your host transfer program should give more information about the meanings of the specific messages.

Win32 error codes reported by EHLLAPI are greater than 300. To determine the Win32 error code, subtract 300 and refer to the *Win32 Programmer's 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=0, and the EOT=c session options. See items 1 and 2 on page 137 plus items 7 and 8 on page 139 for more information.
2. 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 *Personal Communications Version 4.2 Quick Beginnings for Windows 95 and Windows NT.*

Send Key (3)

3270	*5250*	*VT*
Yes	Yes	Yes

The **Send Key** 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 though they were entered by the terminal operator. You can also send all attention identifier (AID) keys such as Enter and so on. All host fields that are input protected or are numeric only must be treated accordingly.

Prerequisite Calls

Connect Presentation Space (1).

Call Parameters

Table 80. Call Parameters

	16-bit	**32-bit**
Function Number	Must be 3.	
Data String	A string of keystrokes, maximum 255. Uppercase and lowercase ASCII characters are represented literally. Function keys and shifted function keys are represented by mnemonics. See “Keyboard Mnemonics” on page 128.	
Length	Length of the source data string. Overridden if in EOT mode.	
PS Postion	NA.	

Return Parameters

Return Code	**Explanation**
0	The keystrokes were sent; status is normal.
1	Your program is not connected to a host session.

Return Code	Explanation
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. The parameters under the **Set Session Parameters** (9) function are related to this function. They are the AUTORESET/NORESET, STRLEN/STREOT, EOT=c, ESC=c, and RETRY/NORETRY session options. See items 1 and 2 on page 137, 9 and 10 on page 140, and 19 on page 143 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 length of the data string must be explicitly defined by the default length parameter, but it can be defined implicitly by the 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 character are always a keycode.

   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 128.

   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 140 (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.

The 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, you should not send a keystroke list that includes plural AID keys.
9. During the @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.

Keyboard Mnemonics

The keyboard mnemonics provide the characters representing the special function keys of the keyboard in the workstation. The abbreviation codes make the mnemonics for special keys easy to remember. An alphabetic key code is used for the most common keys. For example, the **Clear** key is *C*, and the **Tab** key is *T*.

Table 81. Mnemonics with Uppercase Alphabetic Characters

Mnemonic	Meaning	PC/3270	PC400	VT
@B	Left Tab	Yes	Yes	No
@C	Clear	Yes	Yes	No
@D	Delete	Yes	Yes	No
@E	Enter	Yes	Yes	No
@F	Erase EOF	Yes	Yes	No
@H	Help	No	Yes	No
@I	Insert	Yes	Yes	No
@J	Jump (Set Focus)	Yes	Yes	No
@L	Cursor Left	Yes	Yes	Yes
@N	New Line	Yes	Yes	Yes
@O	Space	Yes	Yes	Yes
@P	Print	Yes	Yes	Yes
@R	Reset	Yes	Yes	No
@T	Right Tab	Yes	Yes	Yes
@U	Cursor Up	Yes	Yes	Yes
@V	Cursor Down	Yes	Yes	Yes
@X* **Note:** Not in Windows 3.1	DBCS (Reserved)	Yes	Yes	No
@Y	Caps Lock (No action)	Yes	Yes	No
@Z	Cursor Right	Yes	Yes	Yes

The following table shows the mnemonics using a number or lowercase alphabetic characters.

Table 82. Mnemonics with Numbers or Lowercase Characters

Mnemonic	Meaning	PC/3270	PC400	VT
@0	Home	Yes	Yes	No
@1	PF1/F1	Yes	Yes	No
@2	PF2/F2	Yes	Yes	No
@3	PF3/F3	Yes	Yes	No
@4	PF4/F4	Yes	Yes	No
@5	PF5/F5	Yes	Yes	No
@6	PF6/F6	Yes	Yes	Yes
@7	PF7/F7	Yes	Yes	Yes
@8	PF8/F8	Yes	Yes	Yes
@9	PF9/F9	Yes	Yes	Yes
@a	PF10/F10	Yes	Yes	Yes
@b	PF11/F11	Yes	Yes	Yes
@c	PF12/F12	Yes	Yes	Yes
@d	PF13	Yes	Yes	Yes
@e	PF14	Yes	Yes	Yes
@f	PF15	Yes	Yes	Yes
@g	PF16	Yes	Yes	Yes
@h	PF17	Yes	Yes	Yes
@i	PF18	Yes	Yes	Yes
@j	PF19	Yes	Yes	Yes
@k	PF20	Yes	Yes	Yes
@l	PF21	Yes	Yes	No
@m	PF22	Yes	Yes	No
@n	PF23	Yes	Yes	No
@o	PF24	Yes	Yes	No
@q	End	Yes	Yes	No
@s	ScrLk (No action)	Yes	Yes	Yes
@t	Num Lock (No action)	Yes	Yes	Yes
@u	Page Up	No	Yes	No
@v	Page Down	No	Yes	No
@x	PA1	Yes	Yes	No
@y	PA2	Yes	Yes	No
@z	PA3	Yes	Yes	No

The following table shows the mnemonics using the combination @A and @alphabetic uppercase (A–Z) key.

Table 83 (Page 1 of 2). Mnemonics with @A and @ Uppercase Alphabetic Characters

Mnemonic	Meaning	PC/3270	PC400	VT
@A@C	Test	No	Yes	No
@A@D	Word Delete	Yes	Yes	No
@A@E	Field Exit	Yes	Yes	No

Table 83 (Page 2 of 2). Mnemonics with @A and @ Uppercase Alphabetic Characters				
Mnemonic	**Meaning**	**PC/3270**	**PC400**	**VT**
@A@F	Erase Input	Yes	Yes	No
@A@H	System Request	Yes	Yes	No
@A@I	Insert Toggle	Yes	Yes	No
@A@J	Cursor Select	Yes	Yes	No
@A@L	Cursor Left Fast	Yes	Yes	No
@A@Q	Attention	Yes	Yes	No
@A@R	Device Cancel (Cancels Print Presentation Space)	Yes	Yes	No
@A@T	Print Presentation Space	Yes	Yes	Yes
@A@U	Cursor Up Fast	Yes	Yes	No
@A@V	Cursor Down Fast	Yes	Yes	No
@A@Z	Cursor Right Fast	Yes	Yes	No

The following table shows the mnemonics using the combination @A and @number or @A and @alphabetic lowercase (a–z) key.

Table 84. Mnemonics with @A and @ Lowercase Alphabetic Characters				
Mnemonic	**Meaning**	**PC/3270**	**PC400**	**VT**
@A@9	Reverse Video	Yes	Yes	No
@A@b	Underscore	Yes	No	No
@A@c	Reset Reverse Video	Yes	No	No
@A@d	Red	Yes	No	No
@A@e	Pink	Yes	No	No
@A@f	Green	Yes	No	No
@A@g	Yellow	Yes	No	No
@A@h	Blue	Yes	No	No
@A@i	Turquoise	Yes	No	No
@A@j	White	Yes	No	No
@A@l	Reset Host Colors	Yes	No	No
@A@t	Print (Personal Computer)	Yes	Yes	No
@A@y	Forward Word Tab	Yes	Yes	No
@A@z	Backward Word Tab	Yes	Yes	No

The following table shows the mnemonics using the combination @A and @special character.

Table 85. Mnemonics with @A and @ Alphanumeric (Special) Characters				
Mnemonic	**Meaning**	**PC/3270**	**PC400**	**VT**
@A@–	Field –	No	Yes	No
@A@+	Field +	No	Yes	No
@A@<	Record Backspace	No	Yes	No

The following table shows the mnemonics using the combination @S and @alphabetic lowercase.

Table 86. Mnemonics with @S (Shift) and @ Alphabetic Characters				
Mnemonic	**Meaning**	**PC/3270**	**PC400**	**VT**
@S@E	Print Presentation Space on Host	No	Yes	No
@S@x	Dup	Yes	Yes	No
@S@y	Field Mark	Yes	Yes	No

DBCS Only: The following table shows the mnemonics using the combination @X and @number or @alphabetic lowercase (a–z).

Table 87. Mnemonics Using @X and @Alphabetic Lowercase (For DBCS Only)				
Mnemonic	**Meaning**	**PC/3270**	**PC400**	**VT**
@X@1	Display SO/SI	Yes	Yes	No
@X@5	Generate SO/SI	No	Yes	No
@X@6	Display Attribute	No	Yes	No
@X@7	Forward Character	No	Yes	No
@X@c	Split vertical bar (¦)	No	Yes	No

VT Only: The following table shows the mnemonics using the combination @M and @number or @alphavetic lowercase (a-z)

Table 88 (Page 1 of 4). Mnemonics Using @M, @Q and @Alphabetic Lowercase (For VT Only)				
Mnemonic	**Meaning**	**VT**	**PC/3270**	**PC400**
@M@0	VT Numeric Pad 0	Yes	No	No
@M@1	VT Numeric Pad 1	Yes	No	No
@M@2	VT Numeric Pad 2	Yes	No	No
@M@3	VT Numeric Pad 3	Yes	No	No
@M@4	VT Numeric Pad 4	Yes	No	No
@M@5	VT Numeric Pad 5	Yes	No	No

Table 88 (Page 2 of 4). Mnemonics Using @M, @Q and @Alphabetic Lowercase (For VT Only)

Mnemonic	Meaning	VT	PC/3270	PC400
@M@6	VT Numeric Pad 6	Yes	No	No
@M@7	VT Numeric Pad 7	Yes	No	No
@M@8	VT Numeric Pad 8	Yes	No	No
@M@9	VT Numeric Pad 9	Yes	No	No
@M@-	VT Numeric Pad -	Yes	No	No
@M@,	VT Numeric Pad ,	Yes	No	No
@M@.	VT Numeric Pad .	Yes	No	No
@M@e	VT Numeric Pad Enter	Yes	No	No
@M@f	VT Edit Find	Yes	No	No
@M@i	VT Edit Insert	Yes	No	No
@M@r	VT Edit Remove	Yes	No	No
@M@s	VT Edit Select	Yes	No	No
@M@p	VT Edit Previous Screen	Yes	No	No
@M@n	VT Edit Next Screen	Yes	No	No
@M@a	VT PF1	Yes	No	No
@M@b	VT PF2	Yes	No	No
@M@c	VT PF3	Yes	No	No
@M@d	VT PF4	Yes	No	No
@M@h	VT HOld Screen	Yes	No	No
@M@(space)	Control Code NUL	Yes	No	No
@M@A	Control Code SOH	Yes	No	No
@M@B	Control Code STX	Yes	No	No
@M@C	Control Code ETX	Yes	No	No
@M@D	Control Code EOT	Yes	No	No
@M@E	Control Code ENQ	Yes	No	No
@M@F	Control Code ACK	Yes	No	No
@M@G	Control Code BEL	Yes	No	No

Table 88 (Page 3 of 4). Mnemonics Using @M, @Q and @Alphabetic Lowercase (For VT Only)

Mnemonic	Meaning	VT	PC/3270	PC400
@M@H	Control Code BS	Yes	No	No
@M@I	Control Code HT	Yes	No	No
@M@J	Control Code LF	Yes	No	No
@M@K	Control Code VT	Yes	No	No
@M@L	Control Code FF	Yes	No	No
@M@M	Control Code CR	Yes	No	No
@M@N	Control Code SO	Yes	No	No
@M@O	Control Code SI	Yes	No	No
@M@P	Control Code DLE	Yes	No	No
@M@Q	Control Code DC1	Yes	No	No
@M@R	Control Code DC2	Yes	No	No
@M@S	Control Code DC3	Yes	No	No
@M@T	Control Code DC4	Yes	No	No
@M@U	Control Code NAK	Yes	No	No
@M@V	Control Code SYN	Yes	No	No
@M@W	Control Code ETB	Yes	No	No
@M@X	Control Code CAN	Yes	No	No
@M@Y	Control Code EM	Yes	No	No
@M@Z	Control Code SUB	Yes	No	No
@M@u	Control Code ESC	Yes	No	No
@M@v	Control Code FS	Yes	No	No
@M@w	Control Code GS	Yes	No	No
@M@x	Control Code RS	Yes	No	No
@M@y	Control Code US	Yes	No	No
@M@z	Control Code DEL	Yes	No	No

Table 88 (Page 4 of 4). Mnemonics Using @M, @Q and @Alphabetic Lowercase (For VT Only)

Mnemonic	Meaning	VT	PC/3270	PC400
@Q@A	VT User Defined Key 6	Yes	No	No
@Q@B	VT User Defined Key 7	Yes	No	No
@Q@C	VT User Defined Key 8	Yes	No	No
@Q@D	VT User Defined Key 9	Yes	No	No
@Q@E	VT User Defined Key 10	Yes	No	No
@Q@F	VT User Defined Key 11	Yes	No	No
@Q@G	VT User Defined Key 12	Yes	No	No
@Q@H	VT User Defined Key 13	Yes	No	No
@Q@I	VT User Defined Key 14	Yes	No	No
@Q@J	VT User Defined Key 15	Yes	No	No
@Q@K	VT User Defined Key 16	Yes	No	No
@Q@L	VT User Defined Key 17	Yes	No	No
@Q@M	VT User Defined Key 18	Yes	No	No
@Q@N	VT User Defined Key 19	Yes	No	No
@Q@0	VT User Defined Key 20	Yes	No	No

The following character keys are interpreted as they are.

a–z	!	'	'	<	}
A–Z	$	(	.	>	[
0–9	%	)	/	=	]
~	&	*	:	?	|
#	"	+	;	{	

The following table shows the mnemonics using a special character.

Table 89. Mnemonics with Special Character Keys

Mnemonic	Meaning	PC/3270	PC400	VT
@@	@	Yes	Yes	Yes
@$	Alternate Cursor (The Presentation Manager Interface only)	Yes	Yes	Yes
@<	Backspace	Yes	Yes	Yes

Set Cursor (40)

3270	*5250*	*VT*
Yes	Yes	Yes

The **Set Cursor** function is used to set the position of the cursor within the host presentation space. Before using the **Set Cursor** function, a workstation application must be connected to the host presentation space.

Prerequisite Calls

Connect Presentation Space (1).

Call Parameters

Table 90. Call Parameters

	16-bit	**32-bit**
Function Number	Must be 40	
Data String	NA	
Length	NA	
PS Postion	Desired cursor position in the connected host presentation space	

Return Parameters

Return Code	**Explanation**
0	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 less than 1 or greater than the size of the connected host presentation space was specified.
9	A system error occurred.

Notes on Using This Function

DBCS Only: If the specified cursor is the second byte of the double-byte character, the cursor moves to the first byte of the character and an error code is not returned.

Set Session Parameters (9)

3270	*5250*	*VT*
Yes	Yes	Yes

The **Set Session Parameters** 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 137.

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."

Function Number	Function Name	See Items
1	Connect Presentation Space	12
2	Disconnect Presentation Space	12
3	Send Key	5, 1, 4, 6, 17
4	Wait	12
5	Copy Presentation Space	8, 13, 14, 15, 19, 24
6	Search Presentation Space	5, 1, 2, 7
8	Copy Presentation Space to String	8, 13, 14, 15, 19, 24**
10	Query Sessions	18, 24
15	Copy String to Presentation Space	5, 1, 13, 14, 16, 24
18	Pause	9
30	Search Field	5, 1, 2, 7, 24
33	Copy String to Field	5, 1, 13, 14, 16, 24*
34	Copy Field to String	8, 13, 14, 18, 24
51	Get Key	4
90	Send File	5, 1, 10, 3
91	Receive File	5, 1, 10, 3
Note: DBCS only		

Prerequisite Calls

None.

Call Parameters

Table 91 (Page 1 of 2). Call Parameters

	16-bit	32-bit
Function Number	Must be 9.	

Table 91 (Page 2 of 2). Call Parameters		
	16-bit	**32-bit**
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" on page 137. 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 (the STREOT option is not allowed).	
PS Postion	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 end 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	Convert all field attributes to null characters.

6. The values in the following table affect the **Pause** (18) function.

Value	Explanation
FPAUSE	A full-duration pause lasts for however long 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** **1** 0.5 **2** 1.0 **3** 1.5 **4** 2.0 **5** 2.5 **6** 3.0 **7** 3.5 **8** 4.0 **9** 4.5 **J** 5.0 **K** 5.5 **L** 6.0 **M** 6.5 **N** 7.0

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 attempts 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 the **Connect Presentation Space** (1) and **Disconnect Presentation Space** (2) commands function.

Value	Explanation
CONLOG	Establishes a logical connection between the workstation session and a host session. During Connect, does not jump to the requested presentation space.
CONPHYS	Establishes a physical connection between the workstation session and a host session. During Connect, jumps to the requested presentation space.

12. 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 the **Wait** (4) function, waits up to a minute before timing out on XCLOCK (X []) or XSYSTEM. For the **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 the **Wait** (4) function, waits until XCLOCK (X [])/XSYSTEM clears. This option is not recommended, because control does not return to your application until the host is available. For the **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 the **Wait** (4) function, checks status and returns immediately (no wait). For the **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: Use of NWAIT is recommended.

13. The values in the following table affect **Copy Presentation Space** (5), **Copy Presentation Space to String** (8), **Copy String to Presentation Space** (15), **Copy String to Field** (33), and **Copy Field to String** (34). (5), (8), and (34).

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.

14. The values in the following table affect **Copy Presentation Space** (5), **Copy Presentation Space to String** (8), **Copy String to Presentation Space** (15), **Copy String to Field** (33), and **Copy Field to String** (34). (5), (8), and (34).

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

15. The values in the following table affect **Copy Presentation Space** (5) and **Copy Presentation Space to String** (8) if NOATTRB and NOEAB are specified.

Value	Explanation
BLANK	Convert all unknown values to X'20'.
NOBLANK	Convert all unknown values to X'00'.

The default value is BLANK. If you want to change the default value to NOBLANK, add the following statement in the PCSWIN.INI file (drive:\directory-name\PRIVATE\PCSWIN.INI; drive and directory-name are the drive and the directory where PC/3270 or PC400 was installed.):

```
[API]
NullToBlank=NO
```

16. The values in the following table affect the presentation space size that is returned by the **Query Sessions** (10).

Value	Explanation
CFGSIZE	Returns the configured size of the connected presentation space. This option ignores any override of the configured size by the host.
NOCFGSIZE	Returns the current size of the connected presentation space.

17. The values in the following table affect **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.
NODISPLAY	Do not copy nondisplay fields in the presentation space to the target buffer area. Copy the nondisplay fields to the target buffer as a string of null characters. This allows applications to display the copied buffers in the presentation widow without displaying confidential information, such as passwords.

18. The values in the following table affect **Copy String to Presentation Space** (15) and **Copy String to Field** (33).

Value	Explanation
NOPUTEAB	EAB (or EAD for DBCS) is not contained in the data string of **Copy String to Presentation Space** or **Copy String to Field**.

Value	Explanation
PUTEAB	EAB is contained with character data in the data string of **Copy String to Presentation Space** or **Copy String to Field**.

This option is used for the compatibility with Communications Manager/2. For Communications Manager/2, the data string, which is specified in **Copy String to Presentation Space** or **Copy String to Field**, must be contain EAB (or EAD) with character data when EAB (or EAD) is valid in **Set Session Parameters**. Whereas, for the previous PC/3270 or PC400, the data string specified in these functions must consist of character data only even if EAB (or EAD) is valid. But PC/3270 or PC400 for Windows 95 and Windows NT allows that the data string contains EAB (or EAD) by setting PUTEAB to provide the compatibility with Communications Manager/2.

19. The values in the following table affect the **Send Key** (3) function. Keystrokes are not processed if the keyboard is blocked or in use. The options determine whether the function tries to resend the keystrokes until a 4-minute timeout occurs or if the function returns immediately after determining the keyboard is blocked or in use.

Value	Explanation
RETRY	Continues to attempt to send keystrokes until they are sent or until a 4-minute timeout occurs.
NORETRY	Returns immediately after determining the keyboard is blocked or in use.

20. **DBCS Only:** The values in the following table affect **Copy Presentation Space** (5), **Copy Presentation Space to String** (8), **Copy String to Presentation Space** (15), **Copy String to Field** (33), and **Copy Field to String** (34).

Value	Explanation
NOEAD	DBCS attribute characters are not passed.
EAD	Pass the presentation space data and two attribute characters for the double-byte character set (DBCS). (Users receive 2 bytes for each character other than the data. Therefore, a buffer twice the size of the presentation space must be preallocated.)

21. **DBCS Only:** The values in the following table affect **Copy Presentation Space** (5), **Copy Presentation Space to String** (8), **Copy String to Presentation Space** (15), **Copy String to Field** (33), and **Copy Field to String** (34).

Value	Explanation
NOSO	Pass the presentation space data except Shift-in (SI) and Shift-out (SO) control characters.
SO	Pass the presentation space data including translated SI control character to X'0E' and SO control character to X'0F'. The allocated buffer size depends on the length of the stored data.
SPACESO	Pass the presentation space data including translated SI and SO control characters to X'20' (blank). The allocated buffer size depends on the length of the stored data.

22. The values in the following table affect **Copy Presentation Space** (5), **Copy Presentation Space to String** (8), **Copy String to Presentation Space** (15), **Copy String to Field** (33), **Copy Field to String** (34). **Search Field** (30). and **Query Sessions** (10)

Value	Explanation
EXTEND_PS	5250 emulation supports a presentation space of 24 rows by 80 columns. In some instances, Communications Manager 5250 emulation displays a 25th row. This occurs when either an error message from the host is displayed or when the operator selects the SysReq key. Personal Communications for Windows 95 and Windows NT always displays the same information on the 24th row. By **EXTEND_PS** option, an EHLLAPI application can use the same interface with Communications Manager EHLLAPI and valid presentation space is extended when this condition occurs.
NOEXTEND_PS	The presentation space is not extended when the above condition occurs. This is the default value.

23. The values in the following table affect the **Connect Presentation Space** (1) and **Connect Window Services** (101) functions. The options specify whether an application can or will share the presentation space to which it is connected with another application. Only one of the following values can be specified with each **Set Session Parameter** call.

Value	Explanation
SUPER_WRITE	The application allows other applications that allow sharing and have write access permissions to concurrently connect to the same presentation space. The originating application performs supervisory-type functions but does not create errors for other applications that share the presentation space.
WRITE_SUPER	The application requires write access and allows only supervisory application to concurrently connect to its presentation space. This is the default value.
WRITE_WRITE	The application requires write access and allows partner or other applications with predictable behavior to share the presentation space.
WRITE_READ	The application requires write access and allows other applications that perform read-only functions to share the presentation space. The application is also allowed to copy the presentation space and perform other read-only operations as usual.
WRITE_NONE	The application has exclusive use of the presentation space. No other applications are allowed to share the presentation space, including supervisory applications. The application is allowed to copy the presentation space and perform read-only operations as usual.
READ_WRITE	The application requires only read access to monitor the presentation space and allows other applications that perform read or write, or both, functions to share the presentation space. The application is also allowed to copy the presentation space and perform other read-only operations as usual.

24. The values in the following table allow applications that have presentation space sharing requirements to limit the sharing to a partner application (an application that was developed to work with it).

Value	Explanation
NOKEY	Allows the application to be compatible with existing applications that do not specify the **KEY** parameter.
KEY$*nnnnnnnn*	Uses a keyword to restrict sharing access to the presentation space that it supports. The keyword must be exactly 8 bytes in length.

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 defined:

Return Code	Explanation
0	The session parameters have been set.
2	One or more parameters were not valid.
9	A system error was encountered.

Start Close Intercept (41)

3270	*5250*	*VT*
Yes	Yes	Yes

The **Start Close Intercept** 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

Table 92. Call Parameters

Byte	Definition	
	16-bit	32-bit
Function Number	Must be 41	
Data String	See the following table	
Length	5 or 6	Must be 12
PS Position	NA	

The data string contains the following items.

Table 93 (Page 1 of 2). Calling Data String

Byte		Definition
16-bit	32-bit	
1	1	A 1-character presentation space short name (PSID).
	2–4	Reserved.

Table 93 (Page 2 of 2). Calling Data String		
Byte		**Definition**
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.
6	5	If you request the asynchronous message mode, specify “M”.
	6–8	Reserved.
2–3	9–12	When ‘M’ is specified in position 5 (6 for 16-bit), the window handle of the window that receives the message should be set. The message is a return value of RegisterWindowMessage (“PCSHLL”) (not equal 0).

Return Parameters

This function returns a data string and a return code.

Data String: If asynchronous message mode is not specified in position 5 (6 for 16-bit) and the function is completed successfully, the following data string is returned.

Table 94. Calling Data String		
Byte		**Definition**
16-bit	32-bit	
1	1	A 1-character presentation space short name (PSID).
	2–8	Reserved.
	9–12	4 byte value in which the event object address is returned by EHLLAPI. The application can wait for this event object. (32-bit only).

Data String: If “M” (asynchronous message mode) is specified in position 5 (6 for 16-bit) and the function is completed successfully, the following data string is returned.

Table 95. Calling Data String		
Byte		**Definition**
16-bit	32-bit	
1	1	A 1-character presentation space short name (PSID)
	2–8	Reserved
2–3	9–10	Task ID of asynchronous message mode

Note: If a user selects the close option, 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 26, which shows a close intercept occurred, and LOWORD of lParam parameter contains function number 41.

Return Code: The following codes are defined:

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 is not supported by the emulation program.

Notes on Using This Function

1. The returned event object is in a non-signaled state when the start request is in the signaled state. The event object is in the signaled state each time a close request occurs. To receive notification of multiple close request events, put the event object into the signaled state each time using **SetEvent** or the **Query Close Intercept** (42) function.
2. After using this function, your application program can use the **Query Close Intercept** (42) function to determine when a close request has occurred. The application can wait on the returned event object to determine when the event has occurred.
3. This is not an exclusive call. Multiple applications can request this function for the same short session ID.
4. 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)

3270	*5250*	*VT*
Yes	Yes	Yes

The **Start Host Notification** function begins the process by which your EHLLAPI application program determines if the host presentation space 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

Table 96 (Page 1 of 2). Call Parameters

	16-bit	32-bit
Function Number	Must be 23	
Data String	Preallocated string, see the following table	

Table 96 (Page 2 of 2). Call Parameters		
	16-bit	**32-bit**
Length	6 or 7 implied	16
PS Position	NA	

The calling data string contains these elements:

Table 97. Calling Data String		
Byte		**Definition**
16-bit	32-bit	
1	1	One of the following values: A 1-character presentation space short name (PSID). A blank or null indicating a request for the host-connected host presentation space.
	2–4	Reserved.
2	5	One of the following values: 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 space updates. The character A requesting the asynchronous mode of the notification When A is specified, position 9–12 returns the address of an event object. The event selection character B, O, or P must be placed in position 13. The character M requesting the asynchronous message mode of the notification. When M is specified, the event selection character B, O, or P must be placed in position 13 (7 for 16-bit).
	6–8	Reserved.
3–4	9–19	When “M” is specified in position 5 (2 for 16-bit), the window handle of the window that receives the message should be set. The message is a return value of RegisterWindowMessage (“PCSHLL”) (not equal 0).
5–6	11–12	The data in these positions is ignored by EHLLAPI. However, no error is caused if a program has data in these positions. This data is accepted to provide compatibility with migrating applications.
7	13	One of the following values if position 5 (2 for 16-bit) is A or M: 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.
	14–16	Reserved.

Return Parameters

This function returns a data string and a return code.

Data String: If “A” (asynchronous mode of notification) is specified in position 5 and the function is completed successfully, the following data string is returned:

Table 98. Returned Data String

Byte		Definition
16-bit	32-bit	
1	1	A 1-character presentation space short name (PSID).
	2–8	Reserved.
	9–12	4-byte value in which the event object address is returned by EHLLAPI. The application can wait for this event object. (32-bit only).

Data String: If “M” (asynchronous message mode) is specified in position 5 (2 for 16-bit) and the function is completed successfully, the following data string is returned:

Table 99. Returned Data String

Byte		Definition
16-bit	32-bit	
1	1	A 1-character presentation space short name (PSID)
	2–8	Reserved
3–4	9–10	Task ID of asynchronous message mode

Note: If OIA or presentation space is updated, an application window receives a message. The message is a return value of RegisterWindowMessage (“PCSHLL”). The wParam parameter contains the Task ID returned by the function call. HIWORD of lParam 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.

Return Code: The following codes are defined:

Return Code	Definition
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 4.2 Quick Beginnings for Windows 95 and Windows NT.*
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 of them have been updated.
3. If the application chooses the asynchronous option, it can wait for the returned event object to determine when a host event has occurred.
4. The event object is initially in a non-signaled state and is signaled each time an appropriate event occurs. To receive notification for multiple events, the application must put the event object into the non-signaled state each time it has been signaled using either the **ResetEvent** or the **Query Host Update** (24) function.
5. The next **Start Host Notification** function is not valid while the option is the same as one of the first **Start Host Notification** function calls.
6. This is not an exclusive call. Multiple applications can request this function for the same short session ID.

Start Keystroke Intercept (50)

3270	*5250*	*VT*
Yes	Yes	Yes

The **Start Keystroke Intercept** function allows a workstation 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

Table 100 (Page 1 of 2). Call Parameters

	16-bit	**32-bit**
Function Number	Must be 50	

Table 100 (Page 2 of 2). Call Parameters		
	16-bit	**32-bit**
Data String	See the following table	
Length	6 or 7	Must be 16
PS Position	NA	

Note: **Length**: Number of bytes in the keystroke intercept buffer. EHLLAPI allocates 8 bytes minimum for this buffer.

The calling data string can contain:

Table 101. Calling Data String		
Byte		**Definition**
16-bit	32-bit	
1	1	One of the following values: A specific host presentation space short name (PSID). A blank or null indicating a request for the host-connected host presentation space.
	2–4	Reserved.
2	5	An **option code** character: D for AID keystrokes only. L for all keystrokes. M for requesting the asynchronous message mode of the notification. When M is specified, a code character “D”,or “L” must be placed in position 13 (7 for 16-bit).
	6–8	Reserved.
3–4	9–10	When “M” is specified in position 5 (2 for 16-bit), the window handle of the window that receives the message should be set. The message is a return value of RegeisterWindowMessage (“PCSHLL”) (not equal 0).
5–6	11–12	The data in these positions is ignored by EHLLAPI. However, no error is caused if a program has data in these positions. This data is accepted to provide compatibility with migrating applications.
7	13	One of the following values if position 5 (2 for 16-bit) is M: D for AID keystrokes only. L for all keystrokes.
	14–16	Reserved.

Data String: If “M” (asynchronous message mode) is specified in position 5 (2 for 16-bit) and the function is completed successfully, the following data string is returned:

Table 102 (Page 1 of 2). Returned Data String		
Byte		**Definition**
16-bit	32-bit	

Table 102 (Page 2 of 2). Returned Data String		
Byte		**Definition**
1	1	A 1-character presentation space short name (PSID)
	2–8	Reserved
3–4	9–10	Task ID of asynchronous message mode

Note: If a user sends keystrokes to a session, an application window receives a message. The message is a return value of RegisterWindowMessge ("PCSHLL"). The wParam parameter contains the Task ID returned by the function call. HIWORD of lParam parameter contains return code 0, which shows that the function was successful, and LOWORD of lParam parameter contains function number 50.

Return Parameters

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 3 bytes in the buffer. The next intercepted keystroke is placed in the adjacent 3 bytes. When the **Get Key** (51) function retrieves a keystroke (first-in first-out, or FIFO), the 3 bytes that it occupied are made 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.

   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 to which they were originally intended, and returns only AID keys to the application.
3. Call 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 4.2 Quick Beginnings for Windows 95 and Windows NT.

Start Playing Macro (110)

3270	*5250*	*VT*
Yes	Yes	Yes

The **Start Playing Macro** function invokes a macro. The macro will be executed in the connected session.

Note: This macro must exist in the private directory and no extension should be specified.

Prerequisite Calls

Connect Presentation Space (1).

Call Parameters

Table 103. Call Parameters

	16-bit	**32-bit**
Function Number	Must be 110	
Data String	See the following table	
Length	Length of macro name, plus 3	
PS Position	NA	

Table 104. Calling Data String

Byte		**Definition**
16-bit	32-bit	
1-2		Reserved
3-10		Null terminated macro name

Return Parameters

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)

3270	*5250*	*VT*
Yes	Yes	Yes

The **Stop Close Intercept** function allows the application to turn off the **Start Close Intercept** (41) function. After 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

Table 105. Call Parameters

	16-bit	**32-bit**
Function Number	Must be 43	
Data String	1-character short session ID of the host presentation space	
Length	NA	Must be 4
PS Position	NA	

The calling data string can contain:

Table 106. Calling Data String

Byte		**Definition**
16-bit	32-bit	
1	1	A 1-character presentation space short name (PSID)
	2–4	Reserved

Return Parameters

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** (41) function was issued.
9	A system error occurred.
12	The session stopped.

Stop Host Notification (25)

3270	*5250*	*VT*
Yes	Yes	Yes

The **Stop Host Notification** function disables the capability of the **Query Host Update** (24) function to determine if the host presentation space 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

Table 107. Call Parameters

	16-bit	**32-bit**
Function Number	Must be 121.	
Data String	See the following note.	
Length	1 is implied.	Must be 4.
PS Position	NA.	

The calling data string can contain:

Table 108. Calling Data String

Byte		**Definition**
16-bit	32-bit	
1	1	A 1-character presentation space short name (PSID)
	2–4	Reserved

Note: 1-character short name of the target presentation space ID, or a blank or a null to indicate a request for the host-connected presentation space.

Return Parameters

Return Code	**Definition**
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)

3270	*5250*	*VT*
Yes	Yes	Yes

The **Stop Keystroke Intercept** function ends your application program's ability to intercept keystrokes.

Prerequisite Calls

Start Keystroke Intercept (50).

Call Parameters

Table 109. Call Parameters

	16-bit	**32-bit**
Function Number	Must be 53	
Data String	Short name of the target presentation space (PSID)	
Length	1 is implied	Must be 4
PS Position	NA	

The calling data string can contain:

Table 110. Calling Data String

Byte		**Definition**
16-bit	32-bit	
1	1	A 1-character presentation space short name (PSID)
	2–4	Reserved

Return Parameters

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.

Wait (4)

3270	*5250*	*VT*
Yes	Yes	Yes

The **Wait** function checks the status of the host-connected presentation space. If the session is waiting for a host response (indicated by XCLOCK (X []) or

XSYSTEM), the **Wait** function causes EHLLAPI to wait up to 1 minute to see if the condition clears.

Prerequisite Calls

Connect Presentation Space (1).

Call Parameters

Table 111. Call Parameters

	16-bit	32-bit
Function Number	Must be 4	
Data String	NA	
Length	NA	
PS Position	NA	

Return Parameters

Return Code	Definition
0	The keyboard is unlocked and ready for input.
1	Your application program is not connected to a valid session.
4	Timeout while still in XCLOCK (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 12 on page 140.
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.

Write Structured Fields (127)

3270	*5250*	*VT*
Yes	No	No

The **Write Structured Fields** function allows an application to write structured field data to the host application. If the call specifies “S” (for Synchronous), the application does not receive control until the **Write Structured Fields** function is

completed. If the call specifies “A” (for Asynchronous), the application receives control immediately after the call. If the call specifies “M”, the application receives control immediately after the call. The application may wait for the message. In any case (S, A or M), the application provides the buffer address in which data to the host is to be placed.

For a successful asynchronous completion of this function, the following statements 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 returns the following information:

1. Function request ID
2. Address of the data string from the asynchronous request
3. Length of the data string
4. Return code of the completed function

Prerequisite Calls

Connect for Structured Fields (120) **Allocate Communication Buffer** (123).

Call Parameters

Table 112. Call Parameters

	16-bit	**32-bit**
Function Number	Must be 127	
Data String	See the following table	
Length	8, 10, or 14	Must be 20
PS Position	NA	

The calling data string can contain:

Table 113 (Page 1 of 2). Calling Data String

Byte		**Definition**
16-bit	32-bit	
1	1	A 1-character presentation space short name (PSID).
	2–4	Reserved.

Table 113 (Page 2 of 2). Calling Data String		
Byte		**Definition**
2	5	S or 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, can wait for the event object. M = Asynchronous. Control is returned immediately to the application, can wait for the message.
	6	Reserved.
3–4	7–8	2-byte destination/origin ID.
5–8	9–12	4-byte address of the buffer from which the data is to be written. The buffer must be obtained using the **Allocate Communications Buffer** (123) function.
9–10	13–16	Reserved.
11–12	17–20	When "M" is specified in position 5 (2 for 16-bit), the window handle of the window that receives the message should be set, The message is a return value of RegisterWindowMessage ("PCSHLL") (not equal 0).
13–14		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.

Return Parameters

This function returns a data string and a return code.

Data String: If "A" (asynchronous) is specified in position 5 (2 for 16-bit) and the function is completed successfully, the following data string is returned:

Table 114. Return Parameters		
Byte		**Definition**
9–10	13–14	2-byte Function Request ID. It is used by the **Get Request Completion** (125) function to determine the completion of this function call.
	15–16	Reserved.
	17–20	4-byte value in which the event object address is returned by EHLLAPI. The application can wait for this event object. When the event object is cleared, the application must issue the **Get Request Completion** (125) function call to get results of the **Write Structured Fields** request. (32-bit only).

Note: An event object is returned for each successful asynchronous request. The event object should not be used again. A new event object is returned for each request and is valid for only the duration of that request.

Data String: If "M" (asynchronous message mode) is specified in position 5 (2 for 16-bit) and the function is completed successfully, the following data string is returned:

Table 115. Return Parameters		
Byte		**Definition**
9–10	13–14	2-byte Function Request ID. It is used by the **Get Request Completion** (125) function to determine the completion of this function call.
	15–16	Reserved.
11–12	17–18	Task ID of asynchronous message mode.
	19–20	Reserved.

Note: If the function is completed successfully, an application window receive a message. The message is a return value of RegisterWindowMessage ("PCSHLL"). The 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.

Return Code: The following codes are defined:

Return Code	Explanation
0	The **Write 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.
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. The host is inbound disabled.

Notes on Using This Function

1. Return code 35 will be 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 will be returned if the host is inbound disabled.
4. The EHLLAPI allows for a maximum of 20 asynchronous requests per application to be outstanding. A return code for unavailable resources (RC=11) is returned if more than 20 asynchronous requests are attempted.

5. If you are using IBM Global Network connections, the maximum number of asynchronous requests is 10.

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 of data in the message, the number does not include the buffer header prefix, which contains 8 bytes). This value must be set by the application.
4	1 word	X'0000'
6	1 word	X'0000'
8	8 bytes	Length of the first (or only) structured field message.
10	1 byte	First nonlength byte of the structured field message.
		⋮
m+7	1 byte	Last byte in the structured field message.

Bytes 0 through 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 through *m*+7 are used for the structured field message sent to the host.

Synchronous Requests

When **Write Structured Fields** is requested synchronously (the S option in the data string), control is returned to the application only after the request is satisfied. The application can assume:

The return code is correct.
The data in the communications buffer (read buffer) is correct.
The host is no longer processing the **Write Structured Fields** request.

Asynchronous Requests

When **Write Structured Fields** is requested asynchronously (the A option in the data string), the application *cannot* assume:

The return code is correct.
The data in the communications buffer (write buffer) is correct.
The host is no longer processing the **Write Structured Fields** request.

When requested asynchronously, EHLLAPI returns the following values:

A 16-bit Request ID in positions 13–14 (9–10 for 16-bit) of the data string
The address of a event object in positions 17–20 of the data string.

These are used to complete the asynchronous **Write Structured Fields** call.

The following steps must be completed to determine the outcome of an asynchronous **Write Structured Fields** function call:

If the EHLLAPI return code is not zero, the request failed. No asynchronous request has been made. The application must take appropriate actions before attempting the call again.

If the return code is zero, the application should wait until the event object is in the signaled state by using the **Get Request Completion** (125) function. The event object **Get Request Completion** (125) function) and should not be reused. The event object is valid only for the duration of the **Write Structured Fields** function call through the completion of the **Get Request Completion** (125) function call.

Once the event object is in the signaled state use the returned 16-bit Request ID as the Request ID parameter in a call to the **Get Request Completion** (125) function. The data string returned from the **Get Request Completion** (125) function call contains the final return code of the **Write Structured Fields** function call.

Asynchronous Requests

When **Write Structured Fields** is requested asynchronously (the M option in the data string), the application cannot assume:

The return code is correct
The data in the communications buffer (write buffer) is correct
The host is no longer processing the **Write Structured Fields** request

When requested asynchronously with the M option, EHLLAPI returns the following values:

A 16-bit request ID in positions 13–14 (9–10 for 16-bit) of the data string
Task ID of asynchronous message mode in position 17–18 (11–12 for 16-bit)

These are used to complete the asynchronous **Write Structured Fields** call.

Chapter 4. PCSAPI Functions

PC/3270 Windows modeand PC400 provide another API set, which is defined here and called *PCSAPI*. Whereas EHLLAPI is used to manage the interaction between a workstation application program and host systems after the session is established, the PCSAPI can be used to control the PC/3270 or PC400 session itself.

This chapter describes each individual 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 or C++ for Windows 95 and Windows NT. 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** files with the appropriate object file or libraries.

   You must also link it with the PCSAPI import library, **PCSCALLS.LIB** for 16-bit and **PCSCAL32.LIB** for 32-bit.

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 is:

Function Name
- Function Type
- Parameter Type and Description
- Return Code

Function Type

“Function Type” shows the type of the function in the following format:

TYPE FunctionName(*TYPE Parameter1, ...***)**

Parameter Type and Description

“Parameter Type and Description” lists the type and describes each 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 PCSAPI function.

© Copyright IBM Corp. 1989, 1997

pcsConnect Session

3270	*5250*	*VT*
Yes	Yes	Yes

The **pcsConnectSession** function starts communication with a host session specified by the short session ID. This function is valid for both 3270 and 5250 sessions.

Function Type

BOOL WINAPI pcsConnectSession(*char cShortSessionID.***)**

Parameter Type and Description

char cShortID Presentation space short session ID.

Return Code

Return Code	Meaning
TRUE	Function ended successfully.
FALSE	It means one of the following things: The session has not started. An incorrect session ID was specified. Call failed.

pcsDisconnect Session

3270	*5250*	*VT*
Yes	Yes	Yes

The **pcsDisconnectSession** function stops communication with a host session specified by the short session ID. This function is valid for both 3270 and 5250 sessions.

Function Type

BOOL WINAPI pcsDisconnectSession(*char cShortSessionID***)**

Parameter Type and Description

char cShortID Presentation space short session ID.

Return Code

Return Code	Meaning
TRUE	Function ended successfully.
FALSE	It means one of the following things: The session has not started. An incorrect session ID was specified. Call failed.

pcsQueryEmulatorStatus

3270	*5250*	*VT*
Yes	Yes	Yes

The **pcsQueryEmulatorStatus** function returns the status of the host session specified by the short session ID.

Function Type

ULONG WINAPI pcsQueryEmulatorStatus(*char cShortSessionID***)**

Parameter Type and Description

char cShortSessionID Presentation space short session ID.

Return Code

The return code value should be processed bit-significantly, that is, by either one of the following values or an ORed value out of the following values:

Return Code	Value	Meaning
PCS_SESSION_STARTED	0x00000001	Specified session has started. When this bit is off, the specified session has not started or an incorrect session ID was specified.
PCS_SESSION_ONLINE	0x00000002	Specified session is online (connected). When this bit is off, the specified session is offline (disconnected).
PCS_SESSION_API_ENABLED	0x00000004	API (EHLLAPI, DDE) is enabled on the specified session. If this bit is off, API is disabled on this session.

pcsQuerySessionList

3270	*5250*	*VT*
Yes	Yes	Yes

The **pcsQuerySessionList** function returns a list of all the current host sessions. The application must supply an array of SESSINFO structures as defined in the PCSAPI.H file, and a count of the number of elements in the array. This function fills in the structures with information about each session and returns the number of sessions found.

If the array has fewer elements than there are host sessions, then only the supplied elements of the array are filled in. The function always returns the actual number of sessions, even if the array is too small.

An application can call this function with zero array elements to determine how many sessions exist. A second call can then be made to obtain the session information.

Function Type

ULONG WINAPI pcsQuerySessionList(*ULONG Count, SESSINFO *SessionList***)**

Parameter Type and Description

ULONG Count — Number of elements in the SessionList array.

SESSINFO *SessionList — Pointer to an array of SESSINFO structures as defined in PCSAPI.H.

Return Parameters

Return Code — Total number of PComm sessions. This may be greater than or less than the Count parameter.

SessionList — The array of SESSINFO structures is filled with information about the host sessions. Sessions may be placed in the list in any order. Each SESSINFO structure contains the following fields (defined in PCSAPI32.H)

Name — A union of "char" and "ULONG" which contains the session ID ('A' — 'Z'). In the current implementation of PComm, only the lower byte ("char") is used, the other bytes are returned as zero.

Status — A combination of bit flags which indicate the current status of the session. The flags (PCS_SESSION_*) are defined in the following table.

The status value should be processed bit-significantly, that is, by either one of the following values or an ORed value out of the following values:

Return Code	Meaning
PCS_SESSION_STARTED	The session is running. If this flag is not set, all others are undefined.
PCS_SESSION_ONLINE	The session has established a communications link to the host (e.g. the session is "connected").
PCS_SESSION_API_ENABLED	The session is enabled for programming APIs. If this flag is not set, the EHLLAPI and Host Access Class Library APIs cannot be used on this session.

Example

```
ULONG          NumSessions, i;    // Session counters
SESSINFO     *SessList;            // Array of session information structures
// Find out number of sessions that exist
NumSessions = pcsQuerySessionList (0,NULL);
if (NumSessions == 0) {
   printf("There are no sessions.");
  exit;
}

// Allocate array large enough for all sessions
SessList = (SESSINFO *)malloc(NumSessions * sizeof(SESSINFO));
memset(SessList, 0x00, NumSessions * sizeof(SESSINFO));

// Now read actual session info
pcsQuerySessionList(NumSessions, SessList);

for (i=0; i<NumSessions; i++) {
   if ((SessList[i].Status & PCS_SESSION_STARTED) &&
         (SessList[i].Status & PCS_SESSION_ONLINE))    {

        printf("Session %c is started and connected.",
                  SessList[i].Name.ShortName);
  }
}

exit;
```

pcsQueryWorkstationProfile

3270	*5250*	*VT*
Yes	Yes	Yes

The **pcsQueryWorkstationProfile** function returns the workstation profile name that has been used to invoke 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 WINAPI pcsQueryWorkstationProfile(*char cShortSessionID, PSZ lpBuffer***)**

Parameter Type and Description

char cShortSessionID — Presentation space short session ID.

PSZ lpBuffer — Work buffer to copy a null-terminated workstation profile name. The buffer must be large enough to contain a fully qualified file name.

Return Code

Return Code	Meaning
TRUE	Function ended successfully.
FALSE	It means one of the following things: The session has not started. An incorrect session ID was specified.

pcsStartSession

3270	*5250*	*VT*
Yes	Yes	Yes

The **pcsStartSession** function starts a host session by using a specified workstation profile. A short session ID can also be specified.

Function Type

ULONG WINAPI pcsStartSession(*PSZ lpProfile, char cShortSessionID, USHORT fuCmdShow***)**

Parameter Type and Description

PSZ lpProfile — Path and filename of the profile to load. Path is optional but filename must be specified, (.ws is not assumed).

char cShortSessionID — Presentation space short session ID. Space or NULL indicates the next available session ID.

USHORT fuCmdShow — Specifies how the window is to be displayed. For a list of the acceptable values, see the description of the *nCmdShow* parameter of the **ShowWindow** function of Microsoft Win32 API.

Return Code

Return Code	Value	Meaning
PCS_SUCCESSFUL	0	The function ended successfully.
PCS_INVALID_ID	1	An incorrect session ID was specified.
PCS_USED_ID	2	The specified short session ID is already used.
PCS_INVALID_PROFILE	3	An error was made in specifying the workstation profile, or the window parameter was not valid.
PCS_SYSTEM_ERROR	9	A system error occurred.

pcsStopSession

3270	*5250*	*VT*
Yes	Yes	Yes

The **pcsStopSession** function stops a host session specified by the short session ID.

Function Type

BOOL WINAPI pcsStopSession(*char cShortSessionID, USHORT fuSaveProfile***)**

Parameter Type and Description

char cShortSessionID Presentation space short session ID.

USHORT fuSaveProfile This parameter can be one of the following values:

fuSaveProfile	Value	Meaning
PCS_SAVE_AS_PROFILE	0	Save the profile as specified in the current profile.
PCS_SAVE_ON_EXIT	1	Save the profile on exit.
PCS_NOSAVE_ON_EXIT	2	Do not save the profile on exit.

Return Code

Return Code	Meaning
TRUE	The function ended successfully.
FALSE	It means one of the following things: The session has not started. An incorrect session ID was specified.

Chapter 5. DDE Functions in a 32-Bit Environment

Personal Communications for Windows 95 and Windows NT provides a 32-bit 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 6, "Using DDE Functions with a DDE Client Application" on page 251.

Personal Communications for Windows 95 and Windows NT also supports 16-bit DDE applications. See Appendix F, "DDE Functions in a 16-Bit Environment" on page 319.

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 116 on page 173 describes these levels.

Table 116. 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.	IBM327032
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)

Personal Communications for Windows 95 and Windows NT supports the following applications as Win32 DDE server.

Table 117. Applications for PC/3270 and PC400

Application
IBM327032 IBM525032

The following table lists the topics you can use.

© Copyright IBM Corp. 1989, 1997

Table 118. 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
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 number in this section where more details can be found.

Function	PC/3270	PC400	VT	Page
Find Field	Yes	Yes	Yes	176
Get Keystrokes	Yes	Yes	Yes	178
Get Mouse Input	Yes	Yes	Yes	179
Get Number of Close Requests	Yes	Yes	Yes	182
Get Operator Information Area	Yes	Yes	Yes	183
Get Partial Presentation Space	Yes	Yes	Yes	184
Get Presentation Space	Yes	Yes	Yes	186
Get Session Status	Yes	Yes	Yes	189
Get System Configuration	Yes	Yes	Yes	191
Get System Formats	Yes	Yes	Yes	192
Get System Status	Yes	Yes	Yes	192
Get System SysItems	Yes	Yes	Yes	193
Get System Topics	Yes	Yes	Yes	194
Get Trim Rectangle	Yes	Yes	Yes	195
Initiate Session Conversation	Yes	Yes	Yes	196
Initiate Structured Field Conversation	Yes	No	No	197
Initiate System Conversation	Yes	Yes	Yes	197
Put Data to Presentation Space	Yes	Yes	Yes	198
Search for String	Yes	Yes	Yes	199
Send Keystrokes	Yes	Yes	Yes	200
Session Execute Macro	Yes	Yes	Yes	201
Set Cursor Position	Yes	Yes	Yes	208
Set Mouse Intercept Condition	Yes	Yes	Yes	210
Set Presentation Space Service Condition	Yes	Yes	Yes	212
Set Session Advise Condition	Yes	Yes	Yes	214
Set Structured Field Service Condition	Yes	No	No	216
Start Close Intercept	Yes	Yes	Yes	217
Start Keystroke Intercept	Yes	Yes	Yes	218
Start Mouse Input Intercept	Yes	Yes	Yes	219
Start Read SF	Yes	No	No	222
Start Session Advise	Yes	Yes	Yes	224
Stop Close Intercept	Yes	Yes	Yes	225

Function	PC/3270	PC400	VT	Page
Stop Keystroke Intercept	Yes	Yes	Yes	226
Stop Mouse Input Intercept	Yes	Yes	Yes	227
Stop Read SF	Yes	No	No	227
Stop Session Advise	Yes	Yes	Yes	228
Terminate Session Conversation	Yes	Yes	Yes	229
Terminate Structured Field Conversation	Yes	No	No	229
Terminate System Conversation	Yes	Yes	Yes	229
Write SF	Yes	No	No	230

Refer to “Summary of DDE Functions in a 32-Bit Environment” on page 245 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

Find Field

3270	*5250*	*VT*
Yes	*Yes*	*Yes*

The **Find Field** function returns to the client the information about 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,
              MAKELPARAM(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 types are:

Type	Meaning
bb or T*b*	This field.
P*b*	The previous field, either protected or unprotected.
N*b*	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 *b* symbol represents a required blank.

PC/3270 or PC400 Response

PC/3270 or PC400 returns the following information about 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, one of the following status codes is 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  char   data[(sizeof(DDEDATA)-1)];
   unsigned short uFieldStart;       /* Field start offset
   unsigned short uFieldLength;     /* Field Length
   unsigned char    cAttribute;        /* Attribute character
   unsigned char    ubReserved;        /* *** Reserved ***
} FINDFIELD;

typedef union tagDDE_FINDFIELD
{
  DDEDATA         DDEdata;
  FINDFIELD     DDEfield;
} DDE_FINDFIELD, *lpDDE_FINDFIELD;
```

Get Keystrokes

3270	*5250*	*VT*
Yes	*Yes*	*Yes*

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,
                 MAKELPARAM(cfFormat, aKEYS) );
```

where:

cfFormat Identifies the format for the keystroke information. This must be CF_DSPTEXT.

aKEYS Identifies a 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, one of the following status codes is 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  char   data[(sizeof(DDEDATA)-1)];
   unsigned short uTextType;          /* Type of keystrokes
  unsigned  char   szKeyData_1¨;     /*  Keystrokes
} KEYSTROKE;

typedef union tagDDE_GETKEYSTROKE
{
  DDEDATA         DDEdata;
  KEYSTROKE      DDEkey;
} DDE_GETKEYSTROKE, *lpDDE_GETKEYSTROKE;
```

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:

```
PCS_PURETEXT     0              /* Pure text, no HLLAPI commands
PCS_HLLAPITEXT 1               /* Text, including HLLAPI tokens
```

Get Mouse Input

3270	*5250*	*VT*
Yes	*Yes*	*Yes*

The **Get Mouse Input** function returns the latest mouse input intercepted by the **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,
              MAKELPARAM(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 PC400cannot return the mouse input information, one of the following status codes is 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 char data[(sizeof(DDEDATA)-1)];
   unsigned char PSPos[4];          /* PS Offset - Mouse position
   unsigned char Tab1[1];           /* Tab character
   unsigned char PSRowPos[4];       /* ROW number of Mouse position
   unsigned char Tab2[1];           /* Tab character
   unsigned char PSColPos[4];       /* Col number of Mouse position
   unsigned char Tab3[1];           /* Tab character
   unsigned char PSSize[4];         /* Size of Presentation Space
   unsigned char Tab4[1];           /* Tab character
   unsigned char PSRows[4];         /* Row number of PS
   unsigned char Tab5[1];           /* Tab character
   unsigned char PSCols[4];         /* Column number of PS
   unsigned char Tab6[1];           /* Tab character
   unsigned char Button[1];         /* Type of clicked mouse button
   unsigned char Tab7[1];           /* Tab character
   unsigned char Click[1];          /* Type of clicking
   unsigned char Tab8[1];           /* Tab character
   unsigned char zClickString[1];/* Retrieved string
} MOUSE_CF_TEXT;

typedef union tagDDE_MOUSE_CF_TEXT
{
  DDEDATA          DDEdata;
  MOUSE_CF_TEXT    DDEmouse;
} DDE_MOUSE_CF_TEXT, *lpDDE_MOUSE_CF_TEXT;
```

The following table shows the values in the parameters:

Parameter Name	Meaning	Value
PSPos	PS offset of the position where the mouse was clicked	0 ... (PSSize – 1)
PSRowPos	Row number of the position where the mouse was clicked	0 ... (PSRows – 1)
PSColPos	Column number of the position where the mouse was clicked	0 ... (PSCols – 1)
PSSize	Size of the presentation space	
PSRows	Number of rows of presentation space	

Parameter Name	Meaning	Value
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  char   data[(sizeof(DDEDATA)-1)];
   unsigned short uPSPos;                    /* PS Offset of the Mouse position
   unsigned short uPSRowPos;                  /* ROW number of Mouse position
   unsigned short uPSColPos;                 /* Column number of Mouse position
   unsigned short uPSSize;                   /* Size of Presentation Space
   unsigned short uPSRows;                    /* Row number of PS
   unsigned short uPSCols;                   /* Column number of PS
   unsigned short uButtonType;               /* Type of clicked mouse button
   unsigned short uClickType;                /* Type of clicking
   unsigned char    zClickString[1];          /* Retrieved string
} MOUSE_CF_DSPTEXT;

typedef union tagDDE_MOUSE_CF_DSPTEXT
{
  DDEDATA                DDEdata;
  MOUSE_CF_DSPTEXT   DDEmouse;
} DDE_MOUSE_CF_DSPTEXT, *lpDDE_MOUSE_CF_DSPTEXT;
```

The following table shows the values in the parameters:

Parameter Name	Meaning	Value
uPSPos	PS offset of the position where the mouse was clicked	0 ... (uPSSize – 1)
uPSRowPos	Row number of the position where the mouse was clicked	0 ... (uPSRows – 1)
uPSColPos	Column number of the position where the 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

Parameter Name	Meaning	Value
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

3270	*5250*	*VT*
Yes	*Yes*	*Yes*

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,
              MAKELPARAM(cfFormat, aCLOSE) );
```

where:

cfFormat Identifies the format for the close intercept information. This must be CF_DSPTEXT.

aCLOSE Identifies a 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, one of the following status codes is 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  char   data[(sizeof(DDEDATA)-1)];
   unsigned short uCloseReqCount; /* Number of the close requests.
} CLOSEREQ;

typedef union tagDDE_CLOSEREQ
{
  DDEDATA      DDEdata;
  CLOSEREQ   DDEclose;
} DDE_CLOSEREQ, *lpDDE_CLOSEREQ;
```

Get Operator Information Area

3270	*5250*	*VT*
Yes	*Yes*	*Yes*

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,
              MAKELPARAM(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, one of the following status codes is 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  char   data[(sizeof(DDEDATA)-1)];
  unsigned  char   OIA[80];
} OIADATA;

typedef union tagDDE_OIADATA
{
  DDEDATA     DDEdata;
  OIADATA     DDEoia;
} DDE_OIADATA, *lpDDE_OIADATA;
```

Get Partial Presentation Space

3270	*5250*	*VT*
Yes	*Yes*	*Yes*

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,
              MAKELPARAM(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 the session presentation space 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, one of the following status codes is 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  char   data[(sizeof(DDEDATA)-1)];
   unsigned short uPSPosition;            /* Position of the part of PS
   unsigned short uPSLength;               /* Length of the part of the PS
   unsigned short uPSRows;                  /* PS number of rows
   unsigned short uPSCols;                 /* PS number of columns
   unsigned short uPSOffset;              /* Offset to the presentation space
   unsigned short uFieldCount;           /* Number of fields
   unsigned short uFieldOffset;         /* Offset to the field array
   unsigned char    PSData[1];        /* PS + Field list Array(lpPSFIELDS)
} EPS_CF_DSPTEXT;

typedef union tagDDE_EPS_CF_DSPTEXT
{
  DDEDATA              DDEdata;
  EPS_CF_DSPTEXT   DDEeps;
} DDE_EPS_CF_DSPTEXT, *lpDDE_EPS_CF_DSPTEXT;

# The PSFIELDS structure is replaced with below structure.

typedef struct tagPSFIELDS
{
   unsigned short uFieldStart;       /* Field start offset
   unsigned short uFieldLength;     /* Field Length
   unsigned char    cAttribute;       /* Attribute character
   unsigned char    ubReserved;       /* *** Reserved ***
} PSFIELDS, *lpPSFIELDS;
```

Note: The following examples show how to obtain long pointers to the PS and the PSFIELDS array.

```
lpDDE = (lpDDE_EPS_CF_DSPTEXT)GlobalLock(hData);
lpps = lpDDE->DDEeps.PSData + lpDDE->DDEeps.uPSOffset;
lppsfields = lpDDE->DDEeps.PSData + lpDDE->DDEeps.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  char   data[(sizeof(DDEDATA)-1)];
   unsigned char    PSPOSITION[4];/* Postion of part of the PS
   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 PS
   unsigned char    Tab3[1];          /* Tab character
   unsigned char    PSCOLS[4];       /* Number of Cols in the PS
   unsigned char    Tab4[1];          /* Tab character
  unsigned  char   PS[1];             /*  PS
} EPS_CF_TEXT;

typedef union tagDDE_EPS_CF_TEXT
{
  DDEDATA          DDEdata;
  EPS_CF_TEXT   DDEeps;
} DDE_EPS_CF_TEXT, *lpDDE_EPS_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, *lpFL_CF_TEXT;

typedef struct tagPS_FIELD
{
  unsigned  char   FieldStart[4];
  unsigned  char   TabF1[1];
  unsigned  char   FieldLength[4];
  unsigned  char   TabF2[1];
```

Note: The following examples show how to obtain long pointers to the PS and the PS_FIELD array.

```
lpDDE = (lpDDE_EPS_CF_TEXT)GlobalLock(hData);
lpps = lpDDE->DDEeps.PS;
lpps_field = lpDDE->DDEeps.PS
             +  atoi(lpDDE->DDEeps.PSLENGTH)
              + ((atoi(lpDDE->DDEeps.PSROWS) -1) * 2) // CR/LF
              + 1 + 1 + 4 + 1;    // Tabs + size of field count
```

Get Presentation Space

3270	*5250*	*VT*
Yes	*Yes*	*Yes*

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,
              MAKELPARAM(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 the session presentation space 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, one of the following status codes is 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 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  char   data[(sizeof(DDEDATA)-1)];
   unsigned short uPSSize;              /* Size of the presentation space
   unsigned short uPSRows;               /* PS number of rows
   unsigned short uPSCols;              /* PS number of columns
   unsigned short uPSOffset;           /* Offset to the presentation space
   unsigned short uFieldCount;        /* Number of fields
   unsigned short uFieldOffset;      /* Offset to the field array
   unsigned char    PSData_1¨;          /* PS and Field list Array(lpPSFIELDS)
} PS_CF_DSPTEXT;

typedef union tagDDE_PS_CF_DSPTEXT
{
  DDEDATA              DDEdata;
  PS_CF_DSPTEXT   DDEps;
} DDE_PS_CF_DSPTEXT, *lpDDE_PS_CF_DSPTEXT;

typedef struct tagPSFIELDS
{
   unsigned short uFieldStart;       /* Field start offset
   unsigned short uFieldLength;     /* Field Length
   unsigned char    cAttribute;        /* Attribute character
   unsigned char    ubReserved;        /* *** Reserved ***
} PSFIELDS, *lpPSFIELDS;
```

Note: The following examples show how to obtain long pointers to the PS and the PSFIELDS array.

```
lpDDE = (lpDDE_PS_CF_DSPTEXT)GlobalLock(hData);
lpps = lpDDE->DDEps.PSData + lpDDE->DDEps.uPSOffset;
lppsfields = lpDDE->DDEps.PSData + lpDDE->DDEps.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  char   data[(sizeof(DDEDATA)-1)];
   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;

typedef union tagDDE_PS_CF_TEXT
{
  DDEDATA         DDEdata;
  PS_CF_TEXT   DDEps;
} DDE_PS_CF_TEXT, *lpDDE_PS_CF_TEXT;
```

Following the PS in the buffer is the following additional structure of fields that compose the field list.

```
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, *lpPS_FIELD;
```

Note: The following example shows how to obtain long pointers to the PS and the PS_FIELD array.

```
lpDDE = (lpDDE_PS_CF_TEXT)GlobalLock(hData);
lpps = lpDDE->DDEps.PS;
lpps_field = lpDDE->DDEps.PS
             +  atoi(lpDDE->DDEps.PSSIZE)
              + ((atoi(lpDDE->DDEps.PSROWS) -1) * 2) // CR/LF
              + 1 + 1 + 4 + 1;    // Tabs + size of field count
```

Get Session Status

3270	*5250*	*VT*
Yes	*Yes*	*Yes*

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,
              MAKELPARAM(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, one of the following status codes is 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)
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: 0: The presentation space is unlocked. 4: The presentation space is busy. 5: The presentation space is locked.
Session Window Handle	XXXX	Window handle of the session.

Note:

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

3270	*5250*	*VT*
Yes	*Yes*	*Yes*

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,
              MAKELPARAM(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 is 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
Reserved	XXXXXX	Reserved
Reserved	XXXX	Reserved
Monitor Type	MONO, CGA, EGA, VGA, XGA	Type of the monitor
Country Code	NNNN	Country code used with 3270 or 5250

Get System Formats

3270	*5250*	*VT*
Yes	*Yes*	*Yes*

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,
              MAKELPARAM(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 is 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

3270	*5250*	*VT*
Yes	*Yes*	*Yes*

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,
              MAKELPARAM(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, ASCII	The host system currently supported by Personal Communications .
Emulation Type	3270, 5250, VT	The emulation type supported by Personal Communications .
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 is 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

3270	*5250*	*VT*
Yes	*Yes*	*Yes*

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,
              MAKELPARAM(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 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 is 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

3270	*5250*	*VT*
Yes	*Yes*	*Yes*

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,
              MAKELPARAM(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.

Get Trim Rectangle

3270	*5250*	*VT*
Yes	*Yes*	*Yes*

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,
              MAKELPARAM(cfFormat, aTRIMRECT) );
```

where:

cfFormat Identifies the format for the trim rectangle. This 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)

[1] The actual number of session topics supported depends 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.

If PC/3270 or PC400 cannot return the trim rectangle, one of the following status codes is 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

3270	*5250*	*VT*
Yes	*Yes*	*Yes*

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,
              MAKELPARAM(aIBM327032, aSessionN) );
```

where:

aIBM327032
Identifies the application atom. The string used to create atom aIBM327032 is “IBM327032”. In the PC400, the application atom is aIBM525032 and the string “IBM525032” 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(aIBM327032, 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(aIBM327032, aSystem)
WM_DDE_ACK(aIBM327032, aSessionA)
:
WM_DDE_ACK(aIBM327032, 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

3270	5250	VT
Yes	Yes	Yes

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,
              MAKELPARAM(aIBM327032, aLUN_xxxx) );
```

Where:

aIBM327032
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 returns an acknowledgment message with the following parameter:

```
WM_DDE_ACK(aIBM327032, aLUN_xxxx)
```

Initiate System Conversation

3270	5250	VT
Yes	Yes	Yes

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,
              MAKELPARAM(aIBM327032, aSystem) );
```

where:

aIBM327032
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(aIBM327032, aSystem)

Put Data to Presentation Space

3270	*5250*	*VT*
Yes	*Yes*	*Yes*

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,
             PackDDElParam(WM_DDE_POKE,
              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  char   poke[(sizeof(DDEPOKE)-1)];
   unsigned short uPSStart;                          /* PS Position
   unsigned short uEOFflag;                           /* EOF effective switch
   unsigned char    szStringData[1];                  /* String Data
} PUTSTRING;

typedef union tagDDE_PUTSTRING
{
  DDEPOKE       DDEpoke;
  PUTSTRING   DDEputstring;
} DDE_PUTSTRING, *lpDDE_PUTSTRING;
```

These values are valid at the uEOFflag field:

```
PCS_UNEFFECTEOF 0          /* The string is not truncated at EOF.
PCS_EFFECTEOF     1        /* The string is 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 one of 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

3270	*5250*	*VT*
Yes	*Yes*	*Yes*

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,
              MAKELPARAM(cfFormat, aSTRING) );
```

where:

cfFormat Identifies the format for the search information. This must be 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, one of the following status codes is 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.

Return Code	Explanation
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  char   data[(sizeof(DDEDATA)-1)];
   unsigned short uFieldStart;       /* String start offset
} SEARCH;

typedef union tagSEARCH
{
  DDEDATA   DDEdata;
  SEARCH     DDEsearch;
} DDE_SEARCH, *lpDDE_SEARCH;
```

Send Keystrokes

3270	*5250*	*VT*
Yes	*Yes*	*Yes*

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,
             PackDDElParam(WM_DDE_POKE,
              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  char   poke[(sizeof(DDEPOKE)-1)];
   unsigned short uTextType;                    /* Type of keystrokes
   unsigned short uRetryCount;                  /* Retry count 1 .. 16
  unsigned  char   szKeyData[1];          /*  Keystrokes
} KEYSTROKES;

typedef union tagDDE_SENDKEYSTROKES
{
  DDEPOKE         DDEpoke;
  KEYSTROKES   DDEkeys;
} DDE_SENDKEYSTROKES, *lpDDE_SENDKEYSTROKES;
```

The following key text types are supported:

```
PCS_PURETEXT     0          /* Pure text, no HLLAPI commands
PCS_HLLAPITEXT 1            /* Text, including HLLAPI tokens
```

Note: If the keystrokes are pure text, then specifying PCS_PURETEXT will transfer the keystrokes to the host in the fastest possible manner. If PCS_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 one of 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

3270	*5250*	*VT*
Yes	*Yes*	*Yes*

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,
              (LPARAM)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” on page 202.

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:

KEYBOARD
RECEIVE
SEND
SENDKEY
WAIT
WINDOW

Use a separate DDE_EXECUTE message for each command.

Note:

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:
For OS/400 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 4.2 Reference for Windows 95 and Windows NT* 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:
For OS/400 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 4.2 Reference for Windows 95 and Windows NT* 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:

1.

Text strings are enclosed in double quotation marks.
Macros are prefixed with “macro”.
The argument string for SENDKEY must be 255 characters or fewer.
The following commands are supported.

Table 119 (Page 1 of 4). SENDKEY Command List

Command Name	Token	PC/3270	PC400	VT
Alternate Cursor	alt cursor	Yes	Yes	No
Alternate Viewing Mode	alt view	Yes	Yes	No
Attention	sys attn	Yes	Yes	No
Backspace	backspace	Yes	Yes	Yes
Back Tab	backtab	Yes	Yes	No
Backtab Word	backtab word	Yes	Yes	No
Character Advance	character advance	No	Yes	No
Character Backspace	backspace valid	No	Yes	No
Clear Screen	clear	Yes	Yes	No
Clicker	click	Yes	Yes	No
Color Blue	blue	Yes	No	No
Color Field Inherit	field color	Yes	No	No
Color Green	green	Yes	No	No
Color Pink	pink	Yes	No	No
Color Red	red	Yes	No	No
Color Turquoise	turquoise	Yes	No	No
Color White	white	Yes	No	No
Color Yellow	yellow	Yes	No	No
Cursor Blink	cursor blink	Yes	Yes	No
Cursor Down	down	Yes	Yes	Yes
Cursor Left	left	Yes	Yes	Yes
Cursor Right	right	Yes	Yes	Yes
Cursor Select	cursor select	Yes	Yes	No
Cursor Up	up	Yes	Yes	Yes
Delete Character	delete char	Yes	Yes	No
Delete Word	delete word	Yes	Yes	No
Device Cancel	device cancel	Yes	Yes	No
Dup Field	dup	Yes	Yes	No
Edit Clear	edit-clear	Yes	Yes	Yes
Edit Copy	edit-copy	Yes	Yes	Yes
Edit Cut	edit-cut	Yes	Yes	Yes
Edit Paste	edit-paste	Yes	Yes	Yes
Edit Undo	edit-undo	Yes	Yes	Yes
End Field	end field	Yes	Yes	No

Table 119 (Page 2 of 4). SENDKEY Command List

Command Name	Token	PC/3270	PC400	VT
Enter	enter	Yes	Yes	No
Erase EOF	erase eof	Yes	Yes	No
Erase Field	erase field	Yes	No	No
Erase Input	erase input	Yes	Yes	No
Fast Cursor Down	fast down	Yes	Yes	No
Fast Cursor Left	fast left	Yes	Yes	No
Fast Cursor Right	fast right	Yes	Yes	No
Fast Cursor Up	fast up	Yes	Yes	No
Field Exit	field exit	No	Yes	No
Field Mark	field mark	Yes	Yes	No
Field +	field +	No	Yes	No
Field –	field -	No	Yes	No
Graphic Cursor	+cr	Yes	No	No
Help	help	Yes	Yes	No
Highlighting Field Inherit	field hilight	Yes	No	No
Highlighting Reverse	reverse	Yes	No	No
Highlighting Underscore	underscore	Yes	No	No
Home	home	Yes	Yes	No
Host Print	host print	Yes	No	No
Input	input	Yes	Yes	No
Input nondisplay	input nd	Yes	Yes	No
Insert Toggle	insert	Yes	Yes	No
Lower case	to lower	Yes	No	No
Mark Down	mark down	Yes	Yes	Yes
Mark Left	mark left	Yes	Yes	Yes
Mark Right	mark right	Yes	Yes	Yes
Mark Up	mark up	Yes	Yes	Yes
Move Mark Down	move down	Yes	Yes	Yes
Move Mark Left	move left	Yes	Yes	Yes
Move Mark Right	move right	Yes	Yes	Yes
Move Mark Up	move up	Yes	Yes	Yes
New Line	newline	Yes	Yes	Yes
Next Page	page down	No	Yes	No
Pause 1 second	pause	Yes	Yes	No
Previous Page	page up	No	Yes	No
Print Screen	local copy	Yes	Yes	Yes
Program Attention Key 1	pa1	Yes	No	No
Program Attention Key 2	pa2	Yes	No	No
Program Attention Key 3	pa3	Yes	No	No
Program Function Key 1 ⋮ Program Function Key 5	pf1 ⋮ pf5	Yes ⋮ Yes	Yes ⋮ Yes	No ⋮ No

Table 119 (Page 3 of 4). SENDKEY Command List

Command Name	Token	PC/3270	PC400	VT
Program Function Key 6 ⋮ Program Function Key 20	pf6 ⋮ pf20	Yes ⋮ Yes	Yes ⋮ Yes	Yes ⋮ Yes
Program Function Key 21 ⋮ Program Function Key 24	pf21 ⋮ pf24	Yes ⋮ Yes	Yes ⋮ Yes	No ⋮ No
Quit	quit	Yes	Yes	No
Reset	reset	Yes	Yes	No
Response Time Monitor	rtm	Yes	No	No
Roll Down	roll down	No	Yes	No
Roll Up	roll up	No	Yes	No
Rubout	rubout	Yes	Yes	Yes
Rule	rule	Yes	Yes	Yes
SO/SI Display	so si	Yes	Yes	No
SO/SI Generate	so si generate	No	Yes	No
System Request	sys req	Yes	Yes	No
Tab Field	tab field	Yes	Yes	Yes
Tab Word	tab word	Yes	Yes	No
Test	test request	No	Yes	No
Unmark	unmark	Yes	Yes	Yes
Upper case	to upper	Yes	No	No
Upper/Lower Change	to other	Yes	No	No
Wait for bind	wait app	Yes	Yes	No
Wait for System	wait sys	Yes	Yes	No
Wait transition	wait trn	Yes	Yes	No
Wait while input inh.	wait inp inh	Yes	Yes	No
Window Relocation 1 ⋮ Window Relocation 8	view 1 ⋮ view 8	Yes ⋮ X	Yes ⋮ X	Yes ⋮ X
VT compose	vt compose	No	No	Yes
VT find	vt find	No	No	Yes
VT hold screen	vt hold	No	No	Yes
VT insert here	vt insert	No	No	Yes
VT next screen	vt next	No	No	Yes
VT numeric keypad 0	vt numpad 0	No	No	Yes
VT numeric keypad 1	vt numpad 1	No	No	Yes
VT numeric keypad 2	vt numpad 2	No	No	Yes
VT numeric keypad 3	vt numpad 3	No	No	Yes
VT numeric keypad 4	vt numpad 4	No	No	Yes
VT numeric keypad 5	vt numpad 5	No	No	Yes
VT numeric keypad 6	vt numpad 6	No	No	Yes
VT numeric keypad 7	vt numpad 7	No	No	Yes
VT numeric keypad 8	vt numpad 8	No	No	Yes
VT numeric keypad 9	vt numpad 9	No	No	Yes

Table 119 (Page 4 of 4). SENDKEY Command List				
Command Name	**Token**	**PC/3270**	**PC400**	**VT**
VT numeric keypad -	vt numpad minus	No	No	Yes
VT numeric keypad ,	vt numpad comma	No	No	Yes
VT numeric keypad .	vt numpad period	No	No	Yes
VT numeric keypad enter	vt numpad enter	No	No	Yes
VT PF1	vt pf1	No	No	Yes
VT PF2	vt pf2	No	No	Yes
VT PF3	vt pf3	No	No	Yes
VT PF4	vt pf4	No	No	Yes
VT prev. screen	vt prev	No	No	Yes
VT remove	vt remove	No	No	Yes
VT select	vt select	No	No	Yes
VT user defined function 6	vt user f6	No	No	Yes
VT user defined function 7	vt user f7	No	No	Yes
VT user defined function 8	vt user f8	No	No	Yes
VT user defined function 9	vt user f9	No	No	Yes
VT user defined function 10	vt user f10	No	No	Yes
VT user defined function 11	vt user f11	No	No	Yes
VT user defined function 12	vt 12	No	No	Yes
VT user defined function 13	vt user f13	No	No	Yes
VT user defined function 14	vt user f14	No	No	Yes
VT user defined function 15	vt user f15	No	No	Yes
VT user defined function 16	vt user f16	No	No	Yes
VT user defined function 17	vt user f17	No	No	Yes
VT user defined function 18	vt user f18	No	No	Yes
VT user defined function 19	vt user f19	No	No	Yes
VT user defined function 20	vt user f20	No	No	Yes

Examples:

1. To logon

   [SENDKEY("Logon")]

2. To get reader list

   [SENDKEY("RDRL", enter)]

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("10 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

3270	*5250*	*VT*
Yes	*Yes*	*Yes*

The **Set Cursor Position** function allows the client application to set the cursor position in the session window.

```
PostMessage( hServerWnd,
             WM_DDE_POKE,
             hClientWnd,
             PackDDELParam(WW_DDE_POKE,
              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  char   poke[(sizeof(DDEPOKE)-1)];
   unsigned short uSetCursorType; /* Cursor Set Type
   unsigned short uSetCursor1;       /* Cursor Row or PS Offset
   unsigned short uSetCursor2;       /* Cursor Col
} SETCURSOR;

typedef union tagDDE_SETCURSOR
{
  DDEPOKE       DDEpoke;
  SETCURSOR   DDEsetcursor;
} DDE_SETCURSOR, *lpDDE_SETCURSOR;
```

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 one of 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).

Return Code	Explanation
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

3270	*5250*	*VT*
Yes	*Yes*	*Yes*

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,
             PackDDElParam(WM_DDE_POKE,
              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 char poke[(sizeof(DDEPOKE)-1)];
   unsigned char zMouseCondition[1];
} SETMOUSE_CF_TEXT;

typedef union tagDDE_SETMOUSE_CF_TEXT
{
  DDEPOKE                DDEpoke;
  SETMOUSE_CF_TEXT   DDEcond;
} DDE_SETMOUSE_CF_TEXT, *lpDDE_SETMOUSE_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 char poke[(sizeof(DDEPOKE)-1)];
   BOOL          bLeftButton;      /* Enable intercepting left button
   BOOL          bRightButton;     /* Enable intercepting right button
   BOOL          bMiddleButton;    /* Enable intercepting middle button
   BOOL          bSingleClick;     /* Enable intercepting single click
   BOOL          bDoubleClick;     /* Enable intercepting double click
   BOOL          bRetrieveString; /* Enable intercepting retrieve string
} SETMOUSE_CF_DSPTEXT;

typedef union tagDDE_SETMOUSE_CF_DSPTEXT
{
  DDEPOKE                 DDEpoke;
  SETMOUSE_CF_DSPTEXT   DDEcond;
} DDE_SETMOUSE_CF_DSPTEXT, *lpDDE_SETMOUSE_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 the mouse 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 the Condition parameter is not valid.
6	The specified format is not valid.
9	A system error occurred.

Set Presentation Space Service Condition

3270	*5250*	*VT*
Yes	*Yes*	*Yes*

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 a 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,
             PackDDELParam(WM_DDE_POKE,
              (hData, aEPSCOND) );
```

where:

hData

Identifies a handle to a Windows global memory object containing:

```
typedef struct tagPSSERVCOND
{
  unsigned  char   poke[(sizeof(DDEPOKE)-1)];
   unsigned short uPSStart;                          /* PS Position
   unsigned short uPSLength;                          /* Length of String or PS
   unsigned short uSearchDir;                        /* Direction for search
   unsigned short uEOFflag;                           /* EOF effective switch
   unsigned char    szTargetString[1];               /* Target String
} PSSERVCOND;

typedef union tagDDE_PSSERVCOND
{
  DDEPOKE         DDEpoke;
  PSSERVCOND   DDEcond;
} DDE_PSSERVCOND, *lpDDE_PSSERVCOND;
```

The following values are valid at the uSearchDir field:

```
PCS_SRCHFRWD      0          /* Search forward.
PCS_SRCHBKWD      1          /* Search backward.
```

```
typedef struct tagSETMOUSE_CF_DSPTEXT
{
   unsigned char poke[(sizeof(DDEPOKE)-1)];
   BOOL          bLeftButton;      /* Enable intercepting left button
   BOOL          bRightButton;     /* Enable intercepting right button
   BOOL          bMiddleButton;    /* Enable intercepting middle button
   BOOL          bSingleClick;     /* Enable intercepting single click
   BOOL          bDoubleClick;     /* Enable intercepting double click
   BOOL          bRetrieveString; /* Enable intercepting retrieve string
} SETMOUSE_CF_DSPTEXT;

typedef union tagDDE_SETMOUSE_CF_DSPTEXT
{
  DDEPOKE                 DDEpoke;
  SETMOUSE_CF_DSPTEXT   DDEcond;
} DDE_SETMOUSE_CF_DSPTEXT, *lpDDE_SETMOUSE_CF_DSPTEXT;
```

The following values are valid at the uEOFflag field:

```
PCS_UNEFFECTEOF 0        /* The string is not truncated at EOF.
PCS_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

3270	*5250*	*VT*
Yes	*Yes*	*Yes*

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 224 for more information about starting the advise.

The client sends the following message to set the advise condition.

```
PostMessage( hServerWnd,
             WM_DDE_POKE,
             hClientWnd,
             PackDDElParam(WM_DDE_POKE,
              (hData, aPSCOND) );
```

where:

hData Identifies a handle to a Windows global memory object containing:

```
typedef struct tagSEARCHDATA
{
  unsigned  char   poke[(sizeof(DDEPOKE)-1)];
   unsigned short uPSStart;                          /* PS Position of string
   unsigned short uPSLength;                          /* Length of String
   BOOL                bCaseSensitive;                  /* Case Sensitive TRUE=YES
   unsigned char    SearchString[1]; /* Search String
} SEARCHDATA;

typedef union tagDDE_SEARCHDATA
{
  DDEPOKE        DDEpoke;
  SEARCHDATA   DDEcond;
} DDE_SEARCHDATA, *lpDDE_SEARCHDATA;
```

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 one of 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

3270	5250	VT
Yes	Yes	Yes

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,
             PackDDElParam(WM_DDE_POKE,
              (hData, aSFCOND) );
```

where:

hData Identifies a handle to a Windows global memory object containing:

```
typedef struct tagSFSERVCOND
{
  unsigned  char   poke[(sizeof(DDEPOKE)-1)];
   unsigned short uBufferLength;                  /* Buffer size of Read_SF
   unsigned short uQRLength;                       /* Length of Query Reply dat
   unsigned char    szQueryReply[1];               /* Query Reply data
} SFSERVCOND;

typedef union tagDDE_SFSERVCOND
{
  DDEPOKE         DDEpoke;
  SFSERVCOND   DDEcond;
} DDE_SFSERVCOND, *lpDDE_SFSERVCOND;
```

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 one of 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.

Return Code	Explanation
4	Structured Field Service Condition is already set.
6	The specified format is not valid.
9	A system error occurred.

Start Close Intercept

3270	*5250*	*VT*
Yes	*Yes*	*Yes*

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,
             Pack  DDElParam(WM_DDE_ADVISE,
              (hOptions, aCLOSE) );
```

where:

hOptions Is a handle to a Windows global memory object DDEADVISE structure.

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 Identifies 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 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, aCLOSE)

Return Code	Explanation
1	Close Intercept is already working.
6	The specified format is not valid.

Return Code	Explanation
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  char   data[(sizeof(DDEDATA)-1)];
   unsigned short uCloseReqCount; /* Number of the close requests.
} CLOSEREQ;

typedef union tagDDE_CLOSEREQ
{
  DDEDATA      DDEdata;
  CLOSEREQ   DDEclose;
} DDE_CLOSEREQ, *lpDDE_CLOSEREQ;
```

The DATA messages continue until a Stop Close Intercept message is sent to PC/3270 or PC400.

Start Keystroke Intercept

3270	*5250*	*VT*
Yes	*Yes*	*Yes*

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,
             PackDDElParam(WM_DDE_ADVISE,
              (hOptions, aKEYS) );
```

where:

hOptions Is a handle to a Windows global memory object DDEADVISE structure.

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 Identifies 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 one of 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 is the same format as if the client requested the data item via a DDE_REQUEST.

Start Mouse Input Intercept

3270	*5250*	*VT*
Yes	*Yes*	*Yes*

The **Start Mouse Input Intercept** function allows a client application to intercept mouse input when a terminal operator presses the mouse button on an 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,
             PackDDElParam(WM_DDE_ADVISE,
              (hOptions, aMOUSE) );
```

where:

hOptions Is a handle to a Windows global memory object DDEADVISE structure.

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 Identifies the 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 one of 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 char data[(sizeof(DDEDATA)-1)];
   unsigned char PSPos[4];          /* PS Offset - Mouse position
   unsigned char Tab1[1];           /* Tab character
   unsigned char PSRowPos[4];       /* ROW number of Mouse position
   unsigned char Tab2[1];           /* Tab character
   unsigned char PSColPos[4];       /* Col number of Mouse position
   unsigned char Tab3[1];           /* Tab character
   unsigned char PSSize[4];         /* Size od Presentation Space
   unsigned char Tab4[1];           /* Tab character
   unsigned char PSRows[4[;         /* Row number of PS
   unsigned char Tab5[1];           /* Tab character
   unsigned char PSCols[4];         /* Column number of PS
   unsigned char Tab6[1];           /* Tab character
   unsigned char Button[1];         /* Type of clicked mouse butt n
   unsigned char Tab7[1];           /* Tab character
   unsigned char Click[1];          /* Type of clicking
   unsigned char Tab8[1];           /* Tab character
   unsigned char zClickString[1];/* Retrieved string
} MOUSE_CF_TEXT;

typedef union tagDDE_MOUSE_CF_TEXT
{
  DDEDATA          DDEdata;
  MOUSE_CF_TEXT    DDEmouse;
} DDE_MOUSE_CF_TEXT, *lpDDE_MOUSE_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  char   data[(sizeof(DDEDATA)-1)];
   unsigned short uPSPos;                    /* PS Offset    - Mouse position
   unsigned short uPSRowPos;                  /* ROW number - Mouse position
   unsigned short uPSColPos;                 /* Col number - Mouse position
   unsigned short uPSSize;                   /* Size of Presentation Space
   unsigned short uPSRows;                    /* Row number of PS
   unsigned short uPSCols;                   /* Column number of PS
   unsigned short uButtonType;               /* Type of clicked mouse button
   unsigned short uClickType;                /* Type of clicking
   unsigned char    zClickString[1];         /* Retrieved string
} MOUSE_CF_DSPTEXT;

typedef union tagDDE_MOUSE_CF_DSPTEXT
{
  DDEDATA                DDEdata;
  MOUSE_CF_DSPTEXT   DDEmouse;
} DDE_MOUSE_CF_DSPTEXT, *lpDDE_MOUSE_CF_DSPTEXT;
```

The following table shows the values in the parameters:

Parameter Name	Meaning	Value
uPSPos	PS offset of the position where the mouse was clicked	0 ... (uPSSize – 1)
uPSRowPos	Row number of the position where the mouse was clicked	0 ... (uPSRows – 1)
uPSColPos	Column number of the position where the 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

3270	*5250*	*VT*
Yes	*Yes*	*Yes*

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,
             PackDDElParam(WM_DDE_ADVISE,
              (hOptions, aSF) );
```

where:

hOptions Is a handle to a Windows global memory object DDEADVISE structure.

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 Identifies 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 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, 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 tagMOUSE_CF_DSPTEXT
{
  unsigned  char   data[(sizeof(DDEDATA)-1)];
   unsigned short uPSPos;                        /* PS Offset    - Mouse position
   unsigned short uPSRowPos;                      /* ROW number - Mouse position
   unsigned short uPSColPos;                     /* Col number - Mouse position
   unsigned short uPSSize;                      /* Size of Presentation Space
   unsigned short uPSRows;                       /* Row number of PS
   unsigned short uPSCols;                      /* Column number of PS
   unsigned short uButtonType;                  /* Type of clicked mouse button
   unsigned short uClickType;                   /* Type of clicking
   unsigned char    zClickString[1];            /* Retrieved string
} MOUSE_CF_DSPTEXT;

typedef union tagDDE_MOUSE_CF_DSPTEXT
{
  DDEDATA                DDEdata;
  MOUSE_CF_DSPTEXT   DDEmouse;
} DDE_MOUSE_CF_DSPTEXT, *lpDDE_MOUSE_CF_DSPTEXT;
typedef struct tagSFDATA
{
  unsigned  char   data[(sizeof(DDEDATA)-1)];
   unsigned short uSFLength;                     /* Length of SF data
   unsigned char    szSFData[1];                  /* SF data
} SFDATA;

typedef union tagDDE_SFDATA
{
  DDEDATA   DDEdata;
  SFDATA     DDEsfdata;
} DDE_SFDATA, *lpDDE_SFDATA;
```

The DATA messages continue until a Stop Read SF message is sent to PC/3270.

Start Session Advise

3270	*5250*	*VT*
Yes	*Yes*	*Yes*

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 214.

The client sends the following command to begin a session advise.

```
PostMessage( hServerWnd,
             WM_DDE_ADVISE,
             hClientWnd,
             PackDDElParam(WM_DDE_ADVISE,
              hOptions, aItem) );
```

where:

hOptions Is a handle to a Windows global memory object DDEADVISE structure. This is the structure:

```
typedef struct tagDDEADVISE
{
 unsigned reserved:14;              // Reserved
 unsigned fDeferUpd:1;               // Send notification only
 unsigned fAckReq:1;                  // Client will ACK all notices
 WORD        cfFormat;                    // Clipboard format to use
}   DDEADVISE,  *lpDDEADVISE;
```

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 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, 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 is the same as if the client requested the data item via a DDE_REQUEST.

Stop Close Intercept

3270	*5250*	*VT*
Yes	*Yes*	*Yes*

The **Stop Close Intercept** function ends a client application's ability to intercept close requests. The client sends the following command to perform the **Stop Close Intercept** function.

```
PostMessage( hServerWnd,
             WM_DDE_UNADVISE,
             hClientWnd,
              MAKELPARAM(NULL, aCLOSE) );
```

where:

aCLOSE Identifies 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 one of 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

3270	*5250*	*VT*
Yes	*Yes*	*Yes*

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,
              MAKELPARAM(NULL, aKEYS) );
```

where:

aKEYS Identifies 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 one of 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

3270	*5250*	*VT*
Yes	*Yes*	*Yes*

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,
              MAKELPARAM(NULL, aMOUSE) );
```

where:

aMOUSE Identifies the 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 one of 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

3270	*5250*	*VT*
Yes	*Yes*	*Yes*

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,
              MAKELPARAM(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 one of 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

3270	*5250*	*VT*
Yes	*Yes*	*Yes*

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,
              MAKELPARAM(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 one of 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

3270	5250	VT
Yes	Yes	Yes

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,
             0 );
```

PC/3270 or PC400 Response

PC/3270 or PC400 acknowledges the terminate command with a terminate message:

WM_DDE_TERMINATE

Terminate Structured Field Conversation

3270	5250	VT
Yes	Yes	Yes

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,
             0 );
```

PC/3270 Response

PC/3270 acknowledges the terminate command with a terminate message:

WM_DDE_TERMINATE

Terminate System Conversation

3270	5250	VT
Yes	Yes	Yes

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,
             0  );
```

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 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

3270	*5250*	*VT*
Yes	*Yes*	*Yes*

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,
             PackDDELParam(WM_DDE_POKE,
              hData, aSF) );
```

where:

hData Identifies a handle to a Windows global memory object containing:

```
typedef struct tagWRITESF
{
  unsigned  char   poke[(sizeof(DDEPOKE)-1)];
   unsigned short uSFLength;          /* Length of SF data
   unsigned char    Work[8];           /* Work area
   unsigned char    szSFData[1];       /* SF data
} WRITESF;

typedef union tagDDE_WRITESF
{
  DDEPOKE   DDEpoke;
  WRITESF   DDEwritesf;
} DDE_WRITESF, *lpDDE_WRITESF;
```

aSF Identifies structured field as the item.

PC/3270 Response

PC/3270 receives structured field data and sends it 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 one of 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.

DDE Menu Item API in a 32-Bit Environment

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 in a 32-Bit Environment

To add, delete, and change menu items, the following DDE conversation must take place between the session and DDE menu server application.

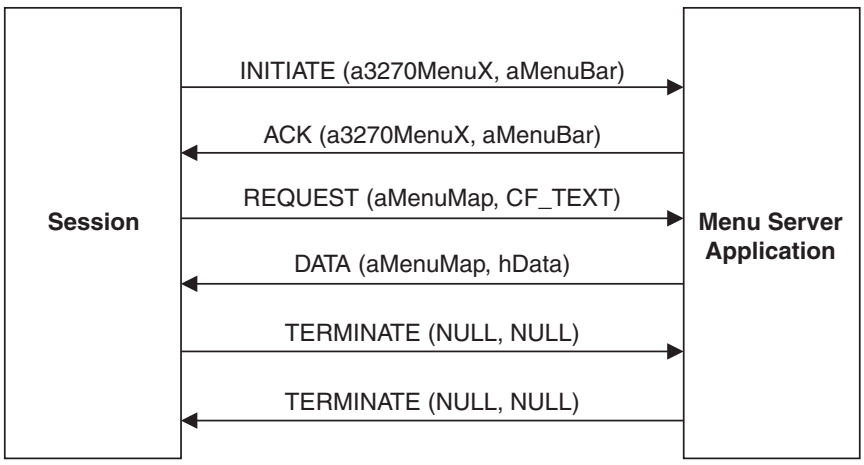


Figure 3. DDE Menu Server Conversation

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 in a 32-Bit Environment

To add, delete, and change menu items, the following DDE conversation must take place between the session and a DDE menu client application.

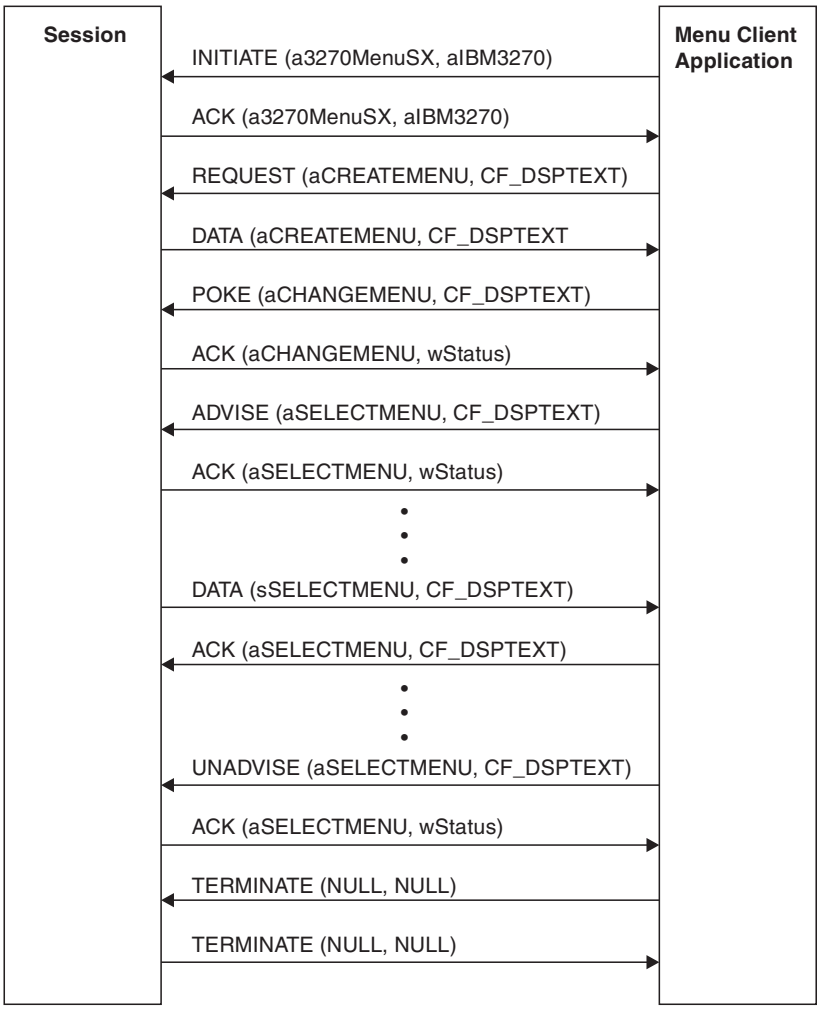


Figure 4. DDE Menu Client Conversation

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 in a 32-Bit Environment

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	233
Create Menu Item	241
Initiate Menu Conversation	242
Start Menu Advise	242
Stop Menu Advise	244
Terminate Menu Conversation	245

Change Menu Item

3270	*5250*	*VT*
Yes	*Yes*	*Yes*

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,
             PackDDElParam(WM_DDE_POKE,
             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  char   poke[(sizeof(DDEPOKE)-1)];
   HWND                hMenu;                          /* Window handle of menu item
   unsigned long    wIDNew;                        /* Menu ID of new menu item
   unsigned short wPosition;                    /* The position of menu item
   unsigned short wOperation;                   /* Specifies the operation
   unsigned short wFlags;                       /* Specifies the options
  unsigned  char   szItemName[1];  /*  String  of  the  item
} CHANGEMENU;

typedef union tagDDE_CHANGEMENU
{
  DDEPOKE         DDEpoke;
  CHANGEMENU   DDEmenu;
} DDE_CHANGEMENU,*lpDDE_CHANGEMENU;
```

The following operations are supported:

```
# MF_APPEND,MF_CHANGE ... MF_BYCOMMANDS are replaced with below commands.
 PCS_INSERT          0x0000     /* Inserts a menu item into a menu.
 PCS_CHANGE          0x0080     /* Modifies a menu item in a menu.
 PCS_APPEND          0x0100     /* Appends a menu item to the end of a menu
 PCS_DELETE          0x0200     /* Deletes a menu item from a menu,
                                /* destroying the menu item.
 PCS_REMOVE          0x1000     /* Removes a menu item from a menu but
                                /* does not destroy the menu item.

 PCS_CHECKED         0x0008     /* Places a checkmark next to the item.
 PCS_DISABLED        0x0002     /* Disables the menu item so that it cannot
                                /* be selected, but does not gray it.
 PCS_ENABLED         0x0000     /* Enables the menu item so that it can be
                                /* selected and restores from its grayed
                                /*  state.
 PCS_GRAYED          0x0001     /* Disables the menu item so that it cannot
                                /* be selected, and grays it.
 PCS_MENUBARBREAK 0x0020     /* Same as PCS_MENUBREAK except that for
                                /* popup menus, separates the new column
                                /* from the old column with a vertical line
 PCS_MENUBREAK       0x0040     /* Places the item on a new line for menu
                                /* bar items. For popup menus, places the
                                /* item in a new column, with no dividing
                                /* line between the columns.
 PCS_SEPARATOR       0x0800     /* Draws a horizontal dividing line. Can
                                /* only be used in a popup menu. This line
                                /* cannot be grayed, disabled, or
                                /* highlighted. The wIDNew and szItemName
                                /* fields are ignored.
 PCS_UNCHAKED        0x0000     /* Does not place a checkmark next to the
                                /* item (default).

 PCS_BYCOMMAND       0x0000     /* Specifies that the nPosition parameter
                                /* gives the menu item control ID number.
                                /* This is the default if neither item
                                /* control ID number. This is the default
                                /* if neither PCS_BYCOMMAND nor
                                /* PCS_POSITION is set.
 PCS_BYPOSITION    0x0400    /* Specifies that the nPosition parameter
                                /* gives the position of the menu item
                                /* to be deleted rather than an ID number.
```

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 the **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 contents 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 the **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 the **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 contents 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 the **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 the **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 the **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 contents 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 the **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 one of 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.

Return code	Explanation
6	The specified format is not valid.
9	A system error occurred.

Create Menu Item

3270	*5250*	*VT*
Yes	*Yes*	*Yes*

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,
             MAKELPARAM(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  char       data[(sizeof(DDEDATA)-1)];
   HWND                hMemuItem;        /* Handle of the menu item
} CREATEMENU;

typedef union tagDDE_CREATEMENU
{
  DDEDATA         DDEdata;
  CREATEMENU   DDEmenu;
} DDE_CREATEMENU,*lpDDE_CREATEMENU;
```

or

WM_DDE_ACK(wStatus,aCREATEMENU)

If PC/3270 or PC400 cannot create a menu item, one of the following status codes is 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

3270	*5250*	*VT*
Yes	*Yes*	*Yes*

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,
             MAKELPARAM(aIBM327032,SN));
```

where:

aIBM327032

Identifies the application atom. The string used to create atom aIBM327032 is IBM327032. In the PC400, the application atom is aIBM525032 and the string IBM525032 is used to create it.

SN

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(aIBM327032,SN)

Start Menu Advise

3270	*5250*	*VT*
Yes	*Yes*	*Yes*

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,
             PackDDElParam(WM_DDE_ADVISE,
             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 0)
 unsigned fAckReq:1;            // Client will ACK all notices
                                    // (Must be 1)
 WORD        cfFormat;              // Always CF_DSPTEXT
}  OPTIONS,FAR  *lpOPTIONS;
```

aSELECTMENU

Identifies 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 is returned to the client with one of 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  char   data[(sizeof(DDEDATA)-1)];
   unsigned short uIDSelected; /* Command ID of the selected menu item
} SELECTMENU;

typedef union tagDDE_SELECTMENU
{
  DDEDATA        DDEdata;
  SELECTMENU   DDEmenu;
} DDE_SELECTMENU,*lpDDE_SELECTMENU;
```

The DATA messages continue until a Stop Menu Advise message is sent to PC/3270 or PC400.

Stop Menu Advise

3270	*5250*	*VT*
Yes	*Yes*	*Yes*

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,
             MAKELPARAM(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 one of 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

3270	5250	VT
Yes	Yes	Yes

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,
             0  );
```

PC/3270 or PC400 Response

PC/3270 or PC400 acknowledges the terminate command with this message:

WM_DDE_TERMINATE

Summary of DDE Functions in a 32-Bit Environment

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, the values that can be used for the variables in the client command, and the server response.

Table 120 (Page 1 of 5). DDE Function Summary in a 32-Bit Environment

function name	client command	server response
Initiate System Conversation(system)	SendMessage(-1, WM_DDE_INITIATE, hClientWnd, MAKELPARAM(aIBM327032,aSystem));	LOWORD/HIWORD to unpack WM_DDE_ACK(aIBM327032,aSystem)
Get System Configuration(system)	PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELPARAM(cfFormat,aSYSCON));	UnPackDDElParam(WM_DDE_DATA,hData,aSYSCON) or UnPackDDElParam(WM_DDE_ACK,wStatus,aSYSCON)
	cfFormat = CF_TEXT	
Get System Formats(system)	PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELPARAM(cfFormat,aFORMATS));	UnPackDDElParam(WM_DDE_DATA,hData,aFORMATS) or UnPackDDElParam(WM_DDE_ACK,wStatus,aFORMATS)
	cfFormat = CF_TEXT	
Get System Status(system)	PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELPARAM(cfFormat,aSTATUS));	UnPackDDElParam(WM_DDE_DATA,hData,aSTATUS) or UnPackDDElParam(WM_DDE_ACK,wStatus,aSTATUS)
	cfFormat = CF_TEXT	
Get System SysItems(system)	PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELPARAM(cfFormat,aSYSITEMS));	UnPackDDElParam(WM_DDE_DATA,hData,aSYSITEMS) or UnPackDDElParam(WM_DDE_ACK,wStatus,aSYSITEMS)
	cfFormat = CF_TEXT	
Get System Topics(system)	PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELPARAM(cfFormat,aTOPICS));	UnPackDDElParam(WM_DDE_DATA,hData,aTOPICS) or UnPackDDElParam(WM_DDE_ACK,wStatus,aTOPICS)
	cfFormat = CF_TEXT	

Table 120 (Page 2 of 5). DDE Function Summary in a 32-Bit Environment

function name	client command	server response
Terminate System Conversation(system)	SendMessage(hServerWnd, WM_DDE_TERMINATE, hClientWnd, 0);	WM_DDE_TERMINATE
Initiate Session Conversation(session)	SendMessage(-1, WM_DDE_INITIATE, hClientWnd, MAKELPARAM(aIBM327032,aSessionN)); N = a session letter A through Z.	LOWORD/HIWORD to unpack WM_DDE_ACK(aIBM327032,aSessionN)
Find Field(session)	PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELPARAM(cfFormat,aFIELD)); cfFormat = CF_DSPTEXT	UnPackDDElParam(WM_DDE_DATA,hData,aFIELD) or UnPackDDElParam(WM_DDE_ACK,wStatus,aFIELD)
Get Keystrokes(session)	PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELPARAM(cfFormat,aKEYS)); cfFormat = CF_DSPTEXT	UnPackDDElParam(WM_DDE_DATA,hData,aKEYS) or UnPackDDElParam(WM_DDE_ACK,wStatus,aKEYS)
Get Mouse Input(session)	PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELPARAM(cfFormat,aMOUSE)); cfFormat = CF_TEXT | CF_DSPTEXT	UnPackDDElParam(WM_DDE_DATA,hData,aMOUSE) or UnPackDDElParam(WM_DDE_ACK,wStatus,aMOUSE)
Get Number of Close Requests(session)	PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELPARAM(cfFormat,aCLOSE)); cfFormat = CF_DSPTEXT	UnPackDDElParam(WM_DDE_DATA,hData,aCLOSE) or UnPackDDElParam(WM_DDE_ACK,wStatus,aCLOSE)
Get Operator Information Area(session)	PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELPARAM(cfFormat,aOIA)); cfFormat = CF_DSPTEXT	UnPackDDElParam(WM_DDE_DATA,hData,aOIA) or UnPackDDElParam(WM_DDE_ACK,wStatus,aOIA)
Get Partial Presentation Space(session)	PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELPARAM(cfFormat,aEPS)); cfFormat = CF_TEXT | CF_DSPTEXT	UnPackDDElParam(WM_DDE_DATA,hData,aEPS) or UnPackDDElParam(WM_DDE_ACK,wStatus,aEPS)
Get Presentation Space(session)	PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELPARAM(cfFormat,aPS)); cfFormat = CF_TEXT | CF_DSPTEXT	UnPackDDElParam(WM_DDE_DATA,hData,aPS) or UnPackDDElParam(WM_DDE_ACK,wStatus,aPS)
Get Session Status(session)	PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELPARAM(cfFormat,aSSTAT)); cfFormat = CF_TEXT	UnPackDDElParam(WM_DDE_DATA,hData,aSSTAT) or UnPackDDElParam(WM_DDE_ACK,wStatus,aSSTAT)
Get Trim Rectangle(session)	PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELPARAM(cfFormat,aTRIMRECT)); cfFormat = CF_TEXT	UnPackDDElParam(WM_DDE_DATA,hData,aTRIMRECT) or UnPackDDElParam(WM_DDE_ACK,wStatus,aTRIMRECT)

Table 120 (Page 3 of 5). DDE Function Summary in a 32-Bit Environment

function name	client command	server response
Put Data to Presentation Space(session)	PostMessage(hServerWnd, WM_DDE_POKE, hClientWnd, PackDDElParam(WM_DDE_POKE, hData,aEPS)); hData = Handle to a global memory object	UnPackDDElParam(WM_DDE_ACK,wStatus,aEPS)
Search for String(session)	PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELPARAM(cfFormat,aSTRING)); cfFormat = CF_DSPTEXT	UnPackDDElParam(WM_DDE_DATA,hData,aSTRING) or UnPackDDElParam(WM_DDE_ACK,wStatus,aSTRING)
Send Keystrokes(session)	PostMessage(hServerWnd, WM_DDE_POKE, hClientWnd, PackDDElParam(WM_DDE_POKE, hData,aKEYS)); hData = Handle to a global memory object	UnPackDDElParam(WM_DDE_ACK,wStatus,aKEYS)
Session Execute Macro(session)	PostMessage(hServerWnd, WM_DDE_EXECUTE, hClientWnd, (LPARAM)hCommands); hCommands = Handle to a global memory object	UnPackDDElParam(WM_DDE_ACK,wStatus, NULL)
Set Cursor Position(session)	PostMessage(hServerWnd, WM_DDE_POKE, hClientWnd, PackDDElParam(WM_DDE_POKE, hData,aSETCURSOR)); hData = Handle to a global memory object	UnPackDDElParam(WM_DDE_ACK,wStatus,aSETCURSOR)
Set Mouse Intercept Condition(session)	PostMessage(hServerWnd, WM_DDE_POKE, hClientWnd, PackDDElParam(WM_DDE_POKE, hData,aMOUSE)); cfFormat = CF_TEXT | CF_DSPTEXT hData = Handle to a global memory object	UnPackDDElParam(WM_DDE_ACK,wStatus,aMOUSE)
Set Presentation Space Service Condition(session)	PostMessage(hServerWnd, WM_DDE_POKE, hClientWnd, PackDDElParam(WM_DDE_POKE, hData,aEPSCOND)); hData = Handle to a global memory object	UnPackDDElParam(WM_DDE_ACK,wStatus,aEPSCOND)
Set Session Advise Condition(session)	PostMessage(hServerWnd, WM_DDE_POKE, hClientWnd, PackDDElParam(WM_DDE_POKE, hData,aPSCOND)); hData = Handle to a global memory object	UnPackDDElParam(WM_DDE_ACK,wStatus,aPSCOND)
Start Close Intercept(session)	SendMessage(hServerWnd, WM_DDE_ADVISE, hClientWnd, PackDDElParam(WM_DDE_ADVISE, hOptions,aCLOSE)); hOptions = Handle to a global memory object	UnPackDDElParam(WM_DDE_ACK,wStatus,aCLOSE) or UnPackDDElParam(WM_DDE_DATA,hData,aCLOSE)

Table 120 (Page 4 of 5). DDE Function Summary in a 32-Bit Environment

function name	client command	server response
Start Keystroke Intercept(session)	SendMessage(hServerWnd, WM_DDE_ADVISE, hClientWnd, PackDDElParam(WM_DDE_ADVISE, hOptions,aKEYS)); hOptions = Handle to a global memory object	UnPackDDElParam(WM_DDE_ACK,wStatus,aKEYS) or UnPackDDElParam(WM_DDE_DATA,hData,aKEYS)
Start Mouse Intercept(session)	SendMessage(hServerWnd, WM_DDE_ADVISE, hClientWnd, PackDDElParam(WM_DDE_ADVISE, hOptions,aMOUSE)); hOptions = Handle to a global memory object	UnPackDDElParam(WM_DDE_ACK,wStatus,aMOUSE) or UnPackDDElParam(WM_DDE_DATA,hData,aMOUSE)
Start Session Advise(session)	PostMessage(hServerWnd, WM_DDE_ADVISE, hClientWnd, PackDDElParam(WM_DDE_ADVISE, hOptions,aItem)); hOptions = Handle to a global memory object aItem = OIA | PS | TRIMRECT	UnPackDDElParam(WM_DDE_ACK,wStatus,aItem) or UnPackDDElParam(WM_DDE_DATA,hData,aItem)
Stop Close Intercept(session)	PostMessage(hServerWnd, WM_DDE_UNADVISE, hClientWnd, MAKELPARAM(NULL,aCLOSE));	UnPackDDElParam(WM_DDE_ACK,wStatus,aCLOSE)
Stop Keystroke Intercept(session)	PostMessage(hServerWnd, WM_DDE_UNADVISE, hClientWnd, MAKELPARAM(NULL,aKEYS));	UnPackDDElParam(WM_DDE_ACK,wStatus,aKEYS)
Stop Mouse Input Intercept(session)	PostMessage(hServerWnd, WM_DDE_UNADVISE, hClientWnd, MAKELPARAM(NULL,aMOUSE));	UnPackDDElParam(WM_DDE_ACK,wStatus,aMOUSE)
Stop Session Advise(session)	PostMessage(hServerWnd, WM_DDE_UNADVISE, hClientWnd, MAKELPARAM(NULL,aItem)); aItem = SysItems | Topics | NULL	UnPackDDElParam(WM_DDE_ACK,wStatus,aItem)
Terminate Session Conversation(session)	SendMessage(hServerWnd, WM_DDE_TERMINATE, hClientWnd, 0);	WM_DDE_TERMINATE
Initiate Structured Field Conversation(structured field)	SendMessage(-1, WM_DDE_INITIATE, hClientWnd, MAKELPARAM(aIBM327032,aLUN_xxxx)); N = a session letter A through Z. xxxx = a user defined string.	LOWORD/HIWORD to unpack WM_DDE_ACK(aIBM327032,aLUN_xxxx)
Terminate Structured Field Conversation(structured field)	SendMessage(hServerWnd, WM_DDE_TERMINATE, hClientWnd, 0);	WM_DDE_TERMINATE
Set Structured Field Service Condition(structured field)	PostMessage(hServerWnd, WM_DDE_POKE, hClientWnd, PackDDElParam(WM_DDE_POKE, hData,aSFCOND)); hData = Handle to a global memory object	UnPackDDElParam(WM_DDE_ACK,wStatus,aSFCOND)

Table 120 (Page 5 of 5). DDE Function Summary in a 32-Bit Environment

function name	client command	server response
Start Read SF(structured field)	PostMessage(hServerWnd, WM_DDE_ADVISE, hClientWnd, PackDDElParam(WM_DDE_ADVISE, hOptions,aSF)); hOptions = Handle to a global memory object	UnPackDDElParam(WM_DDE_ACK,wStatus,aSF) or UnPackDDElParam(WM_DDE_DATA,hData,aSF)
Stop Read SF(structured field)	PostMessage(hServerWnd, WM_DDE_UNADVISE, hClientWnd, MAKELPARAM(NULL,aSF));	UnPackDDElParam(WM_DDE_ACK,wStatus,aSF)
Write SF(structured field)	PostMessage(hServerWnd, WM_DDE_POKE, hClientWnd, PackDDElParam(WM_DDE_POKE, hData,aSF)); hData = Handle to a global memory object	UnPackDDElParam(WM_DDE_ACK,wStatus,aSF)
Initiate Menu Conversation(menu)	SendMessage(-1, WM_DDE_INITIATE, hClientWnd, MAKELPARAM(aIBM327032,a3270MenuSN)); N = a session letter A through Z	LOWORD/HIWORD to unpack WM_DDE_ACK(aIBM327032,a3270MenuSN)
Change Menu Item(menu)	PostMessage(hServerWnd, WM_DDE_POKE, hClientWnd, PackDDElParam(WM_DDE_POKE, hData,aCHANGEMENU)); hData = Handle to a global memory object	UnPackDDElParam(WM_DDE_ACK,wStatus,aCHANGEMENU)
Create Menu Item(menu)	PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELPARAM(cfFormat,aCREATEMENU)); cfFormat = CF_DSPTEXT	UnPackDDElParam(WM_DDE_DATA,hData,aCREATEMENU) or UnPackDDElParam(WM_DDE_ACK,wStatus,aCREATEMENU)
Start Menu Advise(menu)	PostMessage(hServerWnd, WM_DDE_ADVISE, hClientWnd, PackDDElParam(WM_DDE_ADVISE, hOption,aSELECTMENU)); hData = Handle to a global memory object	UnPackDDElParam(WM_DDE_ACK,wStatus,aSELECTMENU) or UnPackDDElParam(WM_DDE_DATA,hData,aSELECTMENU)
Stop Menu Advise(menu)	PostMessage(hServerWnd, WM_DDE_UNADVISE, hClientWnd, MAKELPARAM(NULL,aSELECTMENU));	UnPackDDElParam(WM_DDE_ACK,wStatus,aCLOSE)
Terminate Menu Conversation(menu)	SendMessage(hServerWnd, WM_DDE_TERMINATE, hClientWnd, 0);	WM_DDE_TERMINATE

Chapter 6. Using DDE Functions with a DDE Client Application

Windows 95 and Windows NT allow users to run multiple Windows application programs and to exchange data between Windows application programs. Dynamic data exchange (DDE) 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 responds to the client application.

The client application needs three names (application program name, topic name, and item name) that are recognized by the server application to start the conversation to exchange data. The client application starts a DDE conversation with the server application by specifying the application program and topic names, and defines the exchange data by specifying the 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.

Using the PC/3270 and PC400 DDE Interface

To start a 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 121. Naming Scheme for Data Items

Level	Description	Example
Application	A Windows 95 and Windows NT task, a Windows task, or a specific task of the application. In this book, application programs are PC/3270 or PC400.	IBM327032
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 *IBM327032* or *IBM525032* for 32-bit applications, *IBM3270* or *IBM5250* for 16-bit applications.

Topic Topic specifies the corresponding topic in the application. The following table shows the topics available to the users:

Table 122 (Page 1 of 2). 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

© Copyright IBM Corp. 1989, 1997

Table 122 (Page 2 of 2). Topics for PC/3270 or PC400		
Topic	**Conversation Name**	**Conversation Type**
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 items to be used follow.

System Conversation

To use the PC/3270 or PC400 system DDE interface, do as follows:

1. Start the system conversation.
2. Request system information.
3. Terminate the system conversation.

Starting the DDE System Conversation with PC/3270 or PC400

To use the DDE interface with PC/3270 or PC400, the client application should first start a DDE conversation with PC/3270 or PC400. To start a DDE conversation, specify *IBM327032* or *IBM525032* as an application name for 32-bit applcations; *IBM3270* or *5250* for 16-bit applications and *System* as a topic name in the DDE function (Initiate) in the client application.

Requesting System Information

After starting the 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	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 the DDE System Conversation with PC/3270 or PC400

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 the session conversation.
2. Use DDE functions (Request, Poke, Execute).
3. Terminate the session conversation.

Starting the DDE Session Conversation

To use the DDE interface with PC/3270 or PC400 sessions, the client application should start the DDE conversation with PC/3270 or PC400. To start DDE conversation, specify *IBM327032* or *IBM525032* as an application name for 32-bit applications; *IBM3270* or *IBM5250* for 16-bit applications and *SessionA*, *SessionB*, ..., *SessionZ* as topic names in the DDE function (Initiate) in the client application.

Requesting Data

After starting the DDE conversation, the client application can request data using the DDE function. Session information can be requested by specifying the following item names in the DDE function (Request) in the client application:

Item	Return Data	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 *	Session presentation space of trim rectangle	Get Trim Rectangle
*: Parameter should be added.		

Sending Data to the Emulator Window (Poke)

After starting the DDE conversation, the client application can send data to PC/3270 or PC400 sessions using the DDE functions. The following table shows the valid items for the DDE functions:

Item	Explanation	DDE Function
EPS(pos,bEOF)	Sends an 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 a DDE conversation, the client application can send commands to the PC/3270 or PC400 session window using the DDE functions. Specify the command in the DDE function (Execute) of the client application. See “Session Execute Macro” on page 201 for details.

Terminating the DDE Session 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 the session conversation.
2. Start the **Advise** function.
3. Stop the **Advise** function.
4. Terminate the session conversation.

Starting the DDE Session Conversation (Hot Link)

To use the DDE interface with PC/3270 or PC400 sessions, the client application should start the DDE conversation with PC/3270 or PC400. To start the DDE conversation, specify *IBM327032* or *IBM525032* as an application name for 32-bit applications; *IBM3270* or *IBM5250* for 16-bit applications, and *SessionA*, *SessionB*, ..., *SessionZ* as topic names in the DDE function (Initiate) in the client application.

Starting the Hot Link with the Session Window

After starting the 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 the hot link, which enables the automatic data update:

Item	Explanation	DDE Function
CLOSE	Starts to intercept Window Close requests	Start Close Intercept
KEYS	Starts to intercept keystrokes	Start Keystroke Intercept
PS * OIA TRIMRECT *	Start to retrieve data of PS, OIA, or trim rectangle	Start Session Advise
***:** Parameter should be added.		

Stopping the Hot Link with the Session Window

To terminate the **Advise** function, the client application needs to use the DDE function. Specify the following item names in the DDE function (Unadvise) in the client application to stop the 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 * OIA TRIMRECT *	Stops to Advise function for the session	Stop Session Advise
***:** Use the same parameter that is used when Start Session Advise was called.		

Terminating the DDE Session 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	Topic
Initiate System Conversation	258
Get System Configuration	256
Get System Formats	256
Get System Status	256
Get System SysItems	257
Get System Topics	257
Terminate System Conversation	258

DDE functions for session conversation

Function	Page
Initiate Session Conversation *1	258
Find Field	258
Get Operator Information Area	259
Get Partial Presentation Space	260
Get Presentation Space	261
Get Session Status	261
Get Trim Rectangle	262
Put Data to Presentation Space	263
Search for String	264
Session Execute Macro	264
Set Cursor Position	265
Terminate Session Conversation *2	269

DDE functions for session conversation (hot link)

Function	Page
Initiate Session Conversation (same as *1)	265
Start Close Intercept	266
Start Keystroke Intercept	266
Start Session Advise	267
Stop Close Intercept	268
Stop Keystroke Intercept	269
Stop Session Advise	269
Terminate Session Conversation (same as *2)	269

DDE Functions for System Conversation

The following DDE functions are provided for PC/3270 or PC400 system conversation.

Get System Configuration

The **Get System Configuration** function returns PC/3270 or PC400support-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

The PC/3270 or PC400 system returns the PC/3270 or PC400 system configuration data item.

Returned Information: See "Get System Configuration" on page 191 for details.

If PC/3270 or PC400 do not return the system configuration data item, it may be because:

An incorrect item name was specified.
A system error has occurred.

Get System Formats

The **Get System Formats** function returns a list of Windows Clipboard formats that are 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.

If PC/3270 or PC400 do not return the format data item, it may be because:

An incorrect item name was specified.
A system error has occurred.

Get System Status

The **Get System Status** function returns the status of each configured PC/3270 or PC400session.

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: See "Get System Status" on page 192 for details.

If PC/3270 or PC400 do not return the status data item, it may be because:

An incorrect item name was specified.
A system error has 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

If PC/3270 or PC400 do not return the system data item, it may be because:

An incorrect item name was specified.
A system error has 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

If PC/3270 or PC400 do not return the system data item, it may be because:

An incorrect item name was specified.
A system error has occurred.

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 the PC/3270 or PC400 application name (IBM327032 or IBM525032 for 32-bit applications) or (IBM3270 or IBM5250 for 16-bit applications) and the topic name (System).

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.

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"	"*bb*" or "T*b*" "P*b*" "N*b*" "NP" "NU" "PP" "PU"	This field. The previous field, either protected or unprotected. The next field, either protected or unprotected. The next protected field. The next unprotected field. The previous protected field. The previous unprotected field.

Note: The *b* symbol represents a required blank.

An item in the IBM Personal Communications 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 foregoing 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	Whether the presentation space is formatted or unformatted. If Unformatted is specified, no other field information will be returned.

Field	Returned Information	Explanation
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	Whether the presentation space is formatted or unformatted. If Unformatted is specified, no other field information will be returned.
Field attribute	N	0 = Not 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 = Sighed 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.

If PC/3270 or PC400 do not return the field information, it may be because:

An incorrect item name was specified.
A system error has 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 table shows the OIA information that PC/3270 or PC400 returns:

Offset	Returned Information	Meaning
0	ONLINE LU-LU SSCP-LU	Online, the screen is unowned LU-LU session owns the screen SSCP-LU session owns the screen
9	X X MCHK X CCHK X PCHK X DNW X BUSY X TWAIT X -S X -f X MUCH X UA X -fUA X DEAD X WRONG X SYSTEM X II	Input inhibit Machine check Communication check Program check Device not working Printing Terminal waiting Minus symbol Minus function Input too much Unauthorized operator Unauthorized operator Minus function Incorrect dead key combination Wrong position System waiting Operator input error (PC400)
19	COMM	Communication error
25	MW	Message waiting (PC400)
36	APL	APL (PC/3270)
42	U NUM	Uppercase Numeric
43	A	Caps lock
47	S I	High intensity, operator selectable High intensity, field inherit
49	CS CI	Color, operator selectable Color, field inherit
52	^	Insert mode
61	P-MAL P-PRN P-ASS	Printer malfunction Printer printing Printer assignment

If PC/3270 or PC400 do not return the OIA information, it may be because:

An incorrect item name was specified.
A system error has 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: An item in the IBM Personal Communications 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 foregoing item names.

PC/3270 or PC400 Response

The following table 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	Specified by the number of rows
PS columns	NNNN	Specified by the number of columns
PS	NNNN	PS data starts from this position

If PC/3270 or PC400 do not return the format data items, it may be because:

An incorrect item name was specified.
A system error has 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 table shows the information PC/3270 or PC400 returns:

Field	Returned Information	Explanation
PS size	NNNN	Size of presentation space
PS rows	NNNN	Number of rows
PS columns	NNNN	Number of columns
PS	NNNN	PS data starts from this position

If PC/3270 or PC400 do not return the format data items, it may be because:

An incorrect item name was specified.
A system error has 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 334 for the returned information.

If PC/3270 or PC400 do not return the format data items, it may be because:

An incorrect item name was specified.
A system error has 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: An item in the IBM Personal Communications 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.

If PC/3270 or PC400 do not return the trim rectangle items, it may be because:

An incorrect item name was specified.
A system error has occurred.

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

Put Data to Presentation Space

The **Put Data to Presentation Space** function sends an 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: An item in the IBM Personal Communications 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 foregoing item name.

PC/3270 or PC400 Response

If PC/3270 or PC400 do not accept the string data, it may be because:

An incorrect item name was specified.
The PS position is not valid.
The length is not valid.
The PS input was inhibited.
A system error has 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 a 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: An item in the IBM Personal Communications 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 foregoing item names.

PC/3270 or PC400 Response

PC/3270 or PC400 returns the following information:

Field	Returned Information	Explanation
String start offset	NNNN, None	"None" is returned if the string is not found.

If PC/3270 or PC400 do not return the string start position, it may be because:

An incorrect item name was specified.
The PS position is not valid, or the string is too long.
A system error has 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 347 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 can cause PC/3270 or PC400 not to 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 Rn1Rn2

Parameter Value	Explanation
NNNN	PS offset
Rn1Rn2	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 foregoing item names.

PC/3270 or PC400 Response

If PC/3270 or PC400 do not move the cursor to the specified PS position, it may be because:

An incorrect item name was specified.
The Cursor PS offset is not valid (it must be from 0 to PS size–1).
The Cursor row value is not valid (it must be from 0 to PS row–1).
The Cursor column value is not valid (it must be from 0 to PS column–1).
A system error has 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 Functions 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 foregoing 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 foregoing item name.

PC/3270 or PC400 Response

PC/3270 or PC400 returns the following information:

Field	Returned Information	Explanation
Number of PS close request	NNNN	When a Close request is generated, the client application receives "0001".

If PC/3270 or PC400 do not start to Close intercept, it may be because:

An incorrect item name was specified.

The Close intercept for the session has already started with the same topic name.

A system error has 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 foregoing item name.

PC/3270 or PC400 Response

PC/3270 or PC400 returns the following information:

Field	Returned Information	Explanation
Keys		Refer to Table 119 “SENDKEY Command List” of Chapter 5, DDE Functions in a 32-Bit Environment Session Execute Macro on page 204.

If PC/3270 or PC400 do not start KeyStroke Intercept, it may be because:

An incorrect item name was specified.

The Keystroke Intercept for the session has already started with the same topic name.

A system error has 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
bCaseSen	1 or 0	Case sensitivity **1** Yes **0** No
"string"		Search string Enclose a 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: An item in the IBM Personal Communications 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 foregoing parameter values. The foregoing parameter values can be used when the presentation space 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: An item in the IBM Personal Communications Version 3.1 format is also supported.

The current specified presentation space trim rectangle is used unless the client application specifies the presentation space 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 foregoing item name.

PC/3270 or PC400 Response

Refer to “Get Partial Presentation Space” on page 260, “Get Operator Information Area” on page 259, and “Get Trim Rectangle” on page 262.

If PC/3270 or PC400 do not start Advise, it may be because:

An incorrect item name was specified.
The Advise for the session has already started with the same topic name
A system error has 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 foregoing item name.

PC/3270 or PC400 Response

If PC/3270 or PC400 do not stop Close Intercept, it may be because:

The Advise has not been started.
A system error has 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 foregoing item name.

PC/3270 or PC400 Response

If PC/3270 or PC400 do not stop Keystroke Intercept, it may be because:

An incorrect item name was specified.
The Advise has not been started.
A system error has 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 foregoing item name.

PC/3270 or PC400 Response

If PC/3270 or PC400 do not stop Advise, it may be because:

An incorrect item name was specified.
The Advise has not been started
A system error has 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: This sample program 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 = "|System"
    Text1.LinkMode    =  COLD

     If FunctionComp&= False Then
          MsgBox "Error has occurred", 48, "DDE sample"
    End  If
    Exit  Sub

ErrHandlerInit:
     FunctionComp& = False
    Resume  Next
End Sub

'-- note ----------------------------------------
'
' If you use VisualBasic Version2.0, use "DoEvents"
' function before starting DDE conversation by
' calling "LinkMode" function.
'
'------------------------------------------------
'************************************************
'***                                          ***
'***       Get System Format                  ***
'***                                          ***
'************************************************
'
'     Request a list of PC/3270 or PC400'      Clipboard Format
'
Sub Command2_Click ()
On Error GoTo ErrHandler
     FunctionComp& = True

     Text1.LinkItem = "Formats"
    Text1.LinkRequest

     If FunctionComp&= False Then
          MsgBox "Error has occurred", 48, "DDE sample"
```

```
    End  If
    Exit  Sub

ErrHandler:
     FunctionComp& = False
    Resume  Next
End Sub

'*************************************************
'***                                           ***
'***       Get System Status                   ***
'***                                           ***
'*************************************************
'
'     Requests each PC/3270 or PC400'      Session Status
'
Sub Command2_Click ()
On Error GoTo ErrHandler
     FunctionComp& = True

     Text1.LinkItem = "Status"
    Text1.LinkRequest

     If FunctionComp&= False Then
          MsgBox "Error has occurred", 48, "DDE sample"
    End  If
    Exit  Sub

ErrHandler:
     FunctionComp& = False
    Resume  Next
End Sub

'*************************************************
'***                                           ***
'***       Get System Configuration            ***
'***                                           ***
'*************************************************
'
'     Requests PC/3270 or PC400'      System Configuration Values
'
Sub Command2_Click ()
On Error GoTo ErrHandler
     FunctionComp& = True

     Text1.LinkItem = "SysCon"
    Text1.LinkRequest

     If FunctionComp&= False Then
          MsgBox "Error has 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/3270 or PC400 System Conversation
'
Sub Command2_Click ()
On Error GoTo ErrHandler
     FunctionComp& = True

     Text1.LinkItem = "SysItems"
    Text1.LinkRequest

     If FunctionComp&= False Then
          MsgBox "Error has occurred", 48, "DDE sample"
    End  If
    Exit  Sub

ErrHandler:
     FunctionComp& = False
    Resume  Next
End Sub

'*************************************************
'***                                           ***
'***       Get System Topics                   ***
'***                                           ***
'*************************************************
'
'     Requests a list of PC/3270 or PC400'      Topics
'
Sub Command2_Click ()
On Error GoTo ErrHandler
     FunctionComp& = True

     Text1.LinkItem = "Topics"
    Text1.LinkRequest

     If FunctionComp&= False Then
          MsgBox "Error has 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 = 0
     FunctionComp& = True

     Text1.LinkMode = NONE

     If FunctionComp&= False Then
          MsgBox "Error has 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 = "|SessionA"
     Text1.LinkMode = COLD

     If FunctionComp&= False Then
          MsgBox "Error has occurred", 48, "DDE sample"
    End  If
    Exit  Sub

ErrHandlerInit:
     FunctionComp& = False
    Resume  Next
End Sub

'-- note ----------------------------------------
'
' If you use VisualBasic Version2.0, use "DoEvents"
' function before starting DDE conversation by
' calling "LinkMode" function.
```

```
'
'-------------------------------------------------
'*************************************************
'***                                           ***
'***      Find  Field                          ***
'***                                           ***
'*************************************************
'
'     Requests 100 Field Information of PS Position
'
Sub Command2_Click ()
On Error GoTo ErrHandler
     FunctionComp& = True

     Text1.LinkItem = "FILED(100,""    "")"
    Text1.LinkRequest

     If FunctionComp&= False Then
          MsgBox "Error has 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 has 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 100 to 1000
```

```
'
Sub Command2_Click ()
On Error GoTo ErrHandler
     FunctionComp& = True

     Text1.LinkItem = "EPS(100,1000,1)"
    Text1.LinkRequest

     If FunctionComp&= False Then
          MsgBox "Error has 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 has 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 has 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 has 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 200
'
Sub Command2_Click ()
On Error GoTo ErrHandler
    FunctionComp& = True

    Text1.Text         = "Hello, World!"
    Text1.LinkItem = "EPS(200,1)"
    Text1.LinkPoke

    If FunctionComp&= False Then
        MsgBox "Error has 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 has 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 has 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 has 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 = 0
     FunctionComp& = True

     Text1.LinkMode = NONE

     If FunctionComp&= False Then
          MsgBox "Error has 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 = "|SessionA"
    Text1.LinkItem    =  "CLOSE"
    Text1.LinkMode    =  HOT

     If FunctionComp&= False Then
          MsgBox "Error has occurred", 48, "DDE sample"
    End  If
    Exit  Sub

ErrHandlerInit:
     FunctionComp& = False
    Resume  Next
End Sub

'-- note --------------------------------------------
'
' If you use VisualBasic Version2.0, 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 = "|SessionA"
    Text1.LinkItem    =  "KEYS"
    Text1.LinkMode    =  HOT

     If FunctionComp&= False Then
          MsgBox "Error has occurred", 48, "DDE sample"
    End  If
    Exit  Sub

ErrHandlerInit:
     FunctionComp& = False
    Resume  Next
End Sub
```

```
'-- note --------------------------------------
'
' If you use VisualBasic Version2.0, 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 = "|SessA_PS"
   Text1.LinkItem   =  "PS(1,6,1,""Hello!"")"
   Text1.LinkMode   =  HOT

    If FunctionComp&= False Then
        MsgBox "Error has occurred", 48, "DDE sample"
   End  If

   Exit  Sub

ErrHandlerInit:
    FunctionComp& = False
   Resume  Next
End Sub

'-- note --------------------------------------
'
' If you use VisualBasic Version2.0, 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:R20C40 is changed
'
Sub Command1_Click ()
On Error GoTo ErrHandlerInit
    Dim HOT As Integer
```

```
     HOT = 1
     FunctionComp& = True

    DoEvents
     Text1.LinkTopic = "|SessA_TRIMRECT"
    Text1.LinkItem    =  "TRIMRECT(1,1,20,40)"
    Text1.LinkMode    =  HOT

     If FunctionComp&= False Then
          MsgBox "Error has occurred", 48, "DDE sample"
    End  If
    Exit  Sub

ErrHandlerInit:
     FunctionComp& = False
    Resume  Next
End Sub

'-- note --------------------------------------------
'
' If you use VisualBasic Version2.0, 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 = "|SessA_OIA"
    Text1.LinkItem    =  "OIA"
    Text1.LinkMode    =  HOT

     If FunctionComp&= False Then
          MsgBox "Error has occurred", 48, "DDE sample"
    End  If
    Exit  Sub

ErrHandlerInit:
     FunctionComp& = False
    Resume  Next
End Sub

'-- note --------------------------------------------
'
' If you use VisualBasic Version2.0, 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 = 0
    FunctionComp& = True

    Text1.LinkMode = NONE

    If FunctionComp&= False Then
        MsgBox "Error has occurred", 48, "DDE sample"
   End  If
   Exit  Sub

ErrHandlerTerm:
    FunctionComp& = False
   Resume  Next
End Sub
```

Chapter 7. Server-Requester Programming Interface (SRPI) Support

The Server-Requester Programming Interface (SRPI) is an API that provides access to IBM Enhanced Connectivity Facility (ECF) providing the tools to write SRPI requester programs. SRPI uses a single verb, SEND_REQUEST, to provide a synchronous call-return interface to remote server programs.

PC/3270 SRPI for Windows 95/NT supports the following programs:

32-bit SRPI Requester Program written in C or C++
The existing 16-bit SRPI Requester Program for PC/3270 for Windows 3.1

How to Use SRPI

You can write the application program using the SRPI in C or C++ for Windows 95/NT. To develop a SRPI application, do as follows:

1. Prepare the source code and add the appropriate SRPI calls.
2. Include the header file UUCCPRB.H in the application program.
3. Compile the source code.
4. Link the resultant .OBJ files with the appropriate object file or libraries.

You must also link it with the SRPI import library, PCSCAL32.LIB for 32-bit and PCSCALLS.LIB for 16-bit.

SRPI Compatibility

PC/3270 supports the SRPI function with:

SRPI interface is the same as Personal Communications Version 3.1.

The SRPI interface has the same function calls as the API calls and parameters of CM/2, except that multiple concurrent SRPI requester applications support the same 3270 logical terminal session.

The return codes for errors or successful operations are the same as with CM/2.

The SRPI interface is usable via a host connect of the emulator in all modes (except asynchronous and Control Unit Terminal connection) when the physical connection to the host is through a token ring or a coaxial cable, or through SNA or non-SNA protocols.

If a call is made to the SRPI interface but there is no response from the host due to a communication failure, an associated error is returned to the caller.

SRPI and EHLLAPI are capable of concurrent operations.

SRPI is supported only for C requester.

Server Alias is not supported.

The 3270 screen update notify is not supported.

© Copyright IBM Corp. 1989, 1997

PCSSRPI.DLL is provided to support the existing 16-bit SRPI applications for Personal Communications **PCSSRPI.DLL** converts 16-bit addressing to 32-bit addressing and passes it to PC/3270 SRPI DLL.

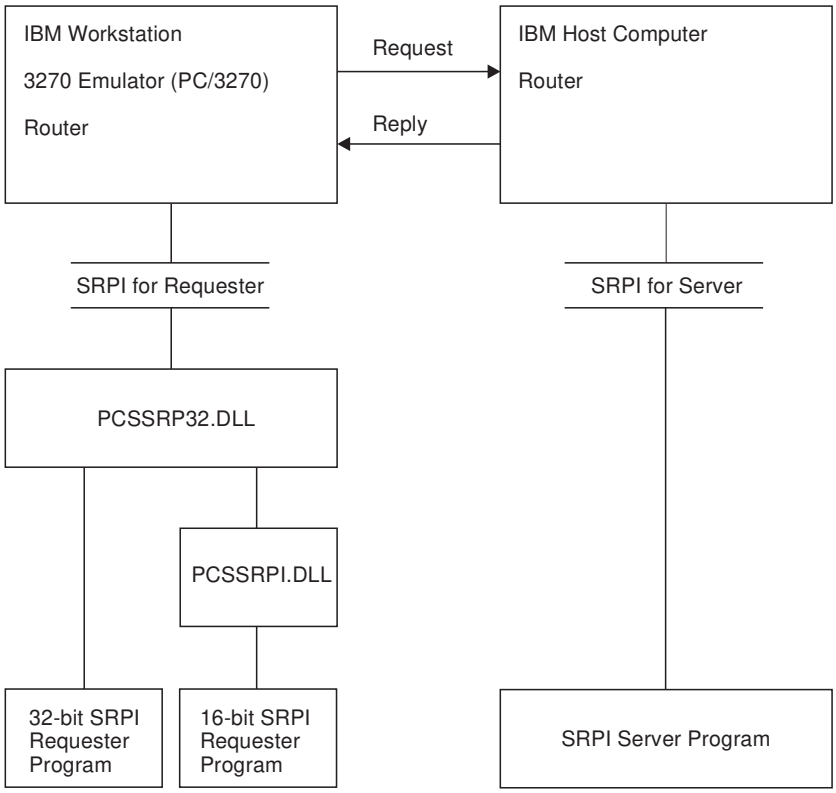


Figure 5. Example of PC/3270 SRPI Requester and Server

Using the Server-Requester Programming Interface

The API between SRPI requesters from the workstation and servers on the host computer is the Server-Requester Programming Interface (SRPI).

Note: For information about a corresponding interface for servers on the IBM host computer, see one of the following publications:

TSO/E Version 2 Guide to the Server-Requester Programming Interface

IBM Programmer's Guide to the Server-Requester Programming Interface for VM/System Product

When used on a workstation, SRPI supports only SRPI requesters. It provides a call-return function for application-to-application communication. Using the **SEND_REQUEST** function, a program on a workstation calls (requests) for service from a partner program on a host computer, which returns (services) the results.

See Figure 6 for an illustration of the workstation and host computer relationship.

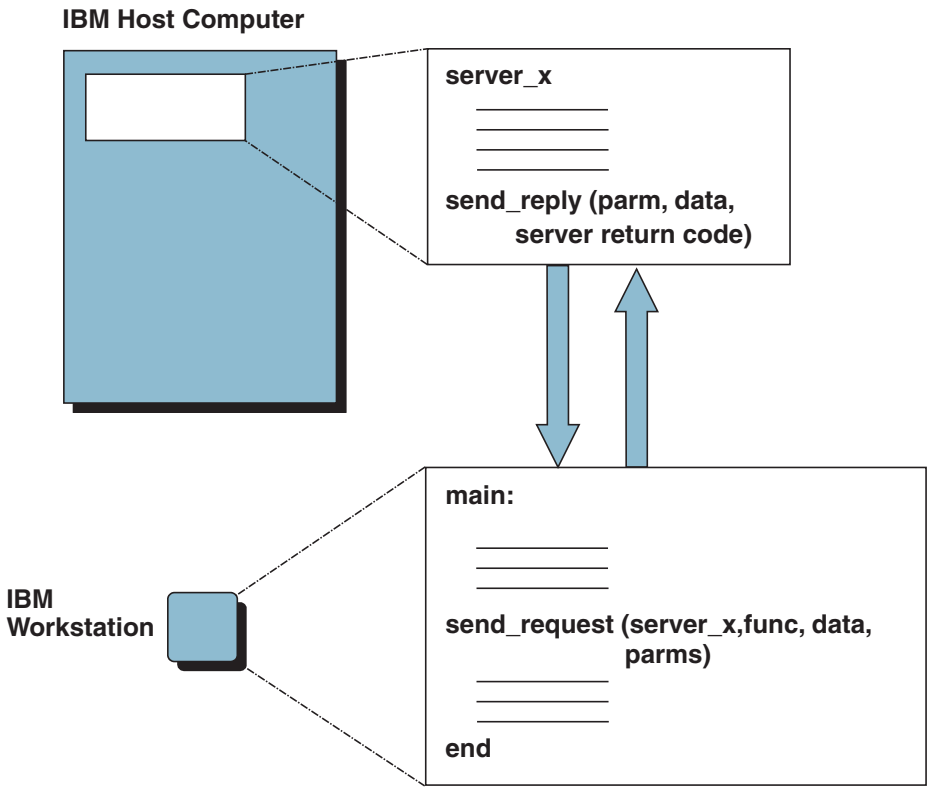


Figure 6. IBM Workstation Requester and IBM Host Computer Server Relationship

Applications use SRPI by issuing the **SEND_REQUEST** verb.

When a workstation SRPI requester issues the **SEND_REQUEST** verb using SRPI:

1. The SRPI router converts the request into a structure that the host computer router recognizes.
2. The SRPI router passes the request to the host computer router, using the appropriate 3270 terminal emulation session.
3. The host computer router passes the request to the appropriate host computer server.
4. The host computer server processes the request and passes a reply back to the host computer router.
5. The host computer router passes the reply back to the SRPI router.
6. The SRPI router converts and returns the reply to the originating SRPI requester application. See Figure 7 for for an illustration of the requester and server flow.

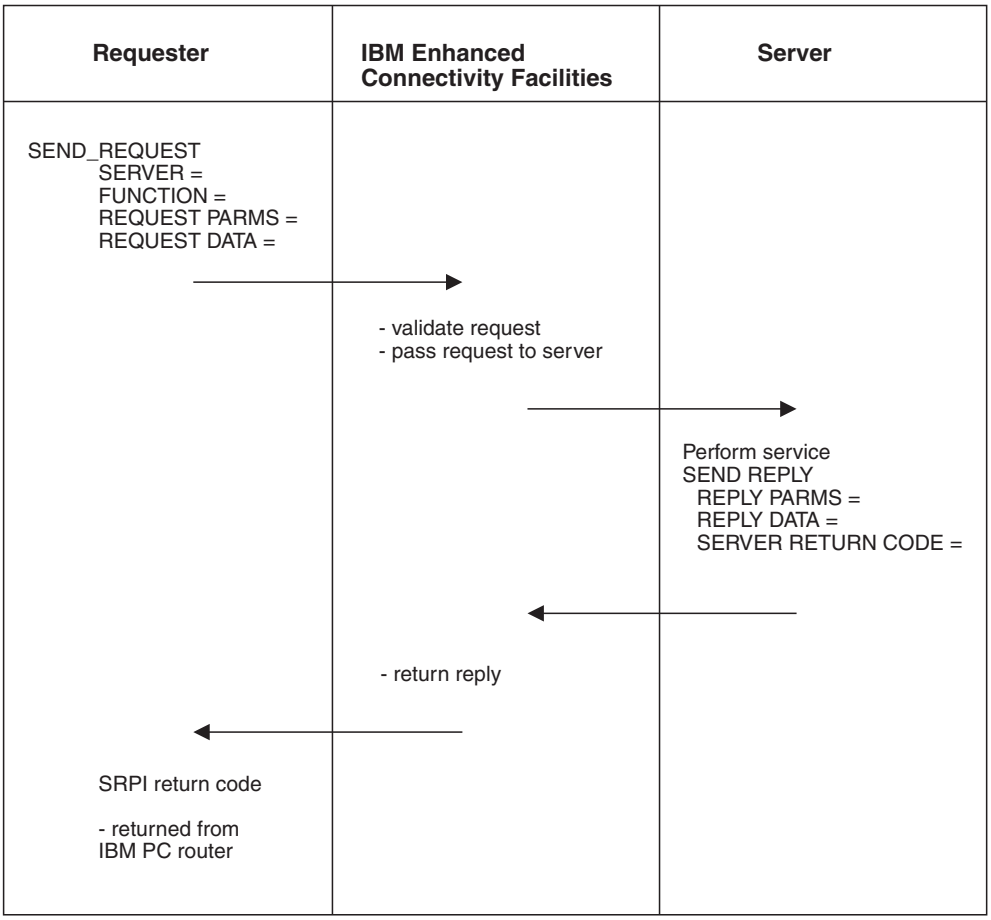


Figure 7. Example of an SRPI Requester and Server Flow

SEND_REQUEST Parameters

The SRPI router sends the request to the host computer using the communication facilities provided by 3270 terminal emulation. SRPI returns control to the SRPI requester with an appropriate return code, optional parameters, and optional data.

The parameters and data associated with the **SEND_REQUEST** function are described in Table 123 on page 287 and in Table 124 on page 288.

Supplied Parameters

Table 123 (Page 1 of 2). Parameters Supplied by the SRPI Requester

Name of Parameter	Required/ Optional	Default Value	Description
Server name	Required	Blanks	The name of the host computer server must be 8 bytes long (PC/ASCII), left-justified, and padded with blanks (X'20'); leading blanks, embedded blanks, and names consisting of all blanks are not valid. The valid PC/ASCII characters are *A* through *Z* (uppercase and lowercase), 0 through 9, $, #, and @. The name is converted to EBCDIC before the request is sent to the host computer.
Function ID	Optional	0	A 2-byte unsigned binary number that specifies the server function being requested. Values of 0 through 65535 are valid for specification by an SRPI requester.
Request parameters length	Optional	0	A 2-byte unsigned binary number that specifies the byte length of the request parameters to be passed to the server. Values of 0 through 32763 are valid. A value of 0 indicates that there are no request parameters to be passed.
Request parameters pointer	Optional	0	The 4-byte address of the parameters, if any, to be passed to the server. A nonzero value in the request parameters length indicates that there are parameters to be passed.
Request data length	Optional	0	A 2-byte unsigned binary number that specifies the byte length of the request data to be passed to the server. Values of 0 through 65535 are valid. A value of 0 indicates that there is no request data to be passed.
Request data pointer	Optional	0	The 4-byte address of the data, if any, to be passed to the server. A nonzero value in the request data length indicates that there is data to be passed.
Reply parameters buffer length	Optional	0	A 2-byte unsigned binary number that specifies the length in bytes of the reply parameter buffer supplied by the SRPI requester. Values of 0 through 32763 are valid. A value of 0 indicates that no reply parameters are expected from the server.
Reply parameters buffer pointer	Optional	0	The 4-byte address of the reply parameter buffer. Its presence is indicated by a nonzero value in the reply parameters buffer length.

Table 123 (Page 2 of 2). Parameters Supplied by the SRPI Requester			
Name of Parameter	**Required/ Optional**	**Default Value**	**Description**
Reply data buffer length	Optional	0	A 2-byte unsigned binary number that specifies the length in bytes of the reply data buffer supplied by the SRPI requester. Values of 0 through 65535 are valid. A value of 0 indicates that no reply data is expected from the server.
Reply data buffer pointer	Optional	0	The 4-byte address of the reply data buffer. A nonzero value in the reply data buffer length indicates that there is reply data to be received.

Returned Parameters

Table 124. Parameters Returned to the SRPI Requester	
Name of Parameter	**Description**
SRPI return code	A 4-byte value that specifies the results of the **SEND_REQUEST** execution. See Appendix E, for a complete description of SRPI return codes.
Server return code	A 4-byte value returned by the server. The contents and meaning of the return status are defined by the requester or the server, but the length of the field is always 32 bits.
Replied parameter length	A 2-byte unsigned binary storage location that specifies the number, in bytes, of parameters returned by the server. Values of 0 through 32763 are valid. A value of 0 indicates that no reply parameters were received from the server.
Replied data length	A 2-byte unsigned binary storage location that specifies the length in bytes of the data returned by the server. Values of 0 through 65535 are valid. A value of 0 indicates that no reply data was received from the server.

Notes:

1. You can set the default values by using the appropriate request record initialization function.
2. The server name is used to route the **SEND_REQUEST** to a 3270 session and to invoke the host server.
3. SRPI requesters and servers determine the contents and meaning of the application data and parameters pointed to by the addresses in the connectivity programming request block (CPRB).

How PC/3270 Applications Use SRPI

A local application running on PC/3270 can issue the **SEND_REQUEST** verb to an application on a connected remote computer. The local application is the SRPI requester and the remote application is the intended server. The SRPI requester can identify a specific function of the server by specifying a function ID.

If the contact is successful, the remote application can provide its services to the SRPI requester. Information on invoking and implementing the **SEND_REQUEST** function follows.

Invoking SEND_REQUEST

When an application invokes **SEND_REQUEST**, it appears to the program that the main routine (the local application) calls a subroutine (the remote application). The programmer who writes the requester application must perform the following tasks:

1. Obtain storage for the connectivity programming request block (CPRB).
2. Initialize the CPRB. This involves setting the default values and completing the application parameters.

   PC/3270 provides initialization routines and macros for each supported language. These initialization facilities insulate the application from the CPRB mapping and call mechanisms.
3. Call the SRPI dynamic link library (DLL) by issuing **SEND_REQUEST**.
4. Validate the SRPI return code received in the CPRB.

The **SEND_REQUEST** function is implemented as a DLL.

Performance Considerations

The size of the data transfer buffers used by the SRPI router to exchange data with the host computer is calculated automatically by PC/3270. If your SRPI requester produces requests that transfer large blocks of data to and from the server, performance might be improved by overriding the data transfer buffer size calculated by PC/3270. This is accomplished by changing the definition of the logical 3270 display terminals used for SRPI.

The data transfer buffer-size override parameter supplied on the Create/Change Logical 3270 Display Terminal window is used to change the buffer size used by SRPI. A value of 0 indicates that PC/3270 calculates the buffer size. Other values (from 1 through 32) specify the buffer size in multiples of 1024 bytes. Be aware that large values (such as 30) might improve SRPI performance at the expense of overall system performance. Note that the data transfer buffer-size override parameter also sets the size of the data transfer buffers used by the File Transfer feature.

Handling the Interrupt (Ctrl+Break) Key

During processing of a **SEND_REQUEST** verb, all signals (except numeric coprocessor signals) are delayed until verb completion. In particular, pressing the Interrupt (Ctrl+Break) key does not cancel a program during execution of a **SEND_REQUEST** verb.

C Requesters

This section is for programmers who want to write a requester in the C language. It describes:

C send_request function
SRPI record definition
Send_request function definition
SRPI return codes

Sample programs are supplied on Personal Communications Version 4.2 for Windows 95 and Windows NTCD-ROM media.

Note: To follow C conventions, the function called **SEND_REQUEST** in other sections is spelled **send_request** in this section.

C send_request Function

The **send_request** parameters are grouped into a single C structure of type UERCPRB. The **init_send_req_parms** function is provided to initialize all **send_request** parameters in the UERCPRB structure to their defaults. This allows the default values to be set once for parameters not used by a requester. The **send_request** function is provided to make synchronous calls to the server program.

The **init_send_req_parms** and **send_request** functions must be linked with your C application. PCSSRP32.DLL for 32-bit interface and PCSSRPI.DLL for 16-bit interface; both of these object files are provided with PC/3270.

The **send_request** function copies the contents of the UERCPRB structure into a connectivity programming request block (CPRB) and calls the PCSSRP32.DLL. After the server has completed its processing, the **send_request** procedure copies the returned parameters from the CPRB into the UERCPRB structure and returns control to the C application.

If the request parameters or data consist of several structures, the application must convert the data or parameters into a single flat structure that consists of a contiguous sequence of bytes that are stored in a buffer. The requesting program must package the request parameters and data in a format recognizable by the server.

UERCPRB is a packed structure. That is, each structure member after the first member is stored at the first available byte.

The memory used for the request parameters can also be used for the reply parameters; the memory used for the request data can also be used for the reply data. The application program must ensure that the reply data and parameters are written into the request data and parameters buffer only when the request data and parameters are no longer needed.

SRPI Record Definition

The UERCPRB record type defines a record passed to the SRPI router using the **send_request** function. The record is defined in an application program by using the #include preprocessor directive to include the UUCCPRB.H file. For the definitions and value ranges of the supplied and returned parameters, see “Supplied Parameters” on page 286 and “Returned Parameters” on page 288.

SRPI Return Codes

See Appendix E, “SRPI Return Codes” on page 311 for the SRPI return codes.

Appendix A. 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/origin ID (DOID) self-defining parameter (SDP) for the structured field support to work and be reliable. 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, 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. Here are some of them:

Table 125. 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. 1989, 1997

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 self-defining parameter 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 126. DDM Application Name Self-Defining Parameter

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 consists of 8 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 Controls self-defining parameter indicates that the PCLK Protocol Controls structured field, ID = X'1013', can be used for both inbound and outbound in data streams destined to or from the DDM auxiliary device processor.

Table 127. DDM PCLK Auxiliary Device Self-Defining Parameter

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 (self-defining parameter) combinations possible. Not all of the combinations are shown.

Table 128. Base DDM Query Reply Format with Name and Direct Access Self-Defining Parameters

Offset	Length	Content	Meaning
0	1 word	Length	Length of structure (includes self-defining parameters)
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/origin ID assigned by the subsystem

The self-defining parameters 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 129. Base DDM Query Reply Format with Direct Access and Name Self-Defining Parameters

Offset	Length	Content	Meaning
0	1 word	Length	Length of structure (includes self-defining parameters)
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/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 self-defining parameters 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 details.

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 auxiliary devices 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 130. IBM Auxiliary Device Base Format with Direct Access Self-Defining Parameter

Offset	Length	Content	Meaning
0–1	1 word	Length	Length of structure (includes self-defining parameters)
2	1 byte	X'81'	Query reply ID
3	1 byte	X'9E'	IBM auxiliary device reply
4	1 byte BIT 0 1–7	FLAGS QUERY B'1' RES	Reserved Read Part (Query, Query List) Auxiliary device supports Query 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/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/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 an 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 131 on page 296.

Direct Access Self-Defining Parameter

The direct access 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 131. IBM Auxiliary Device Direct Access Self-Defining Parameter

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.

PCLK Protocol Controls Self-Defining Parameter

The presence of the PCLK protocol controls self-defining parameter indicates that the PCLK protocol controls 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 132. IBM Auxiliary Device PCLK Self-Defining Parameter

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 133. OEM Auxiliary Device Base Format with Direct Access Self-Defining Parameter			
Offset	**Length**	**Content**	**Meaning**
0–1	1 word	Length	Length of structure (includes self-defining parameters)
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/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 134.

Direct Access Self-Defining Parameter

The direct access self-defining parameter provides the ID for use in the destination/origin structured field in the direct access of the OEM auxiliary device.

Table 134. OEM Auxiliary Device Direct Access Self-Defining Parameter			
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.

PCLK Protocol Controls Self-Defining Parameter

The presence of the PCLK protocol controls self-defining parameter indicates that the PCLK Protocol Controls 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 135. IBM Auxiliary Device PCLK Self-Defining Parameter			
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 136. CPR Query Reply Buffer Format			
Offset	**Length**	**Content**	**Meaning**
0	1 word	Length	Length of structure (includes self-defining parameters)
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– (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/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 137. IBM Product-Defined Query Reply Base Format

Offset	Length	Content	Meaning
0–1	1 word	Length	Length of structure (includes self-defining parameters)
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 138. 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 *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 139.

Direct Access Self-Defining Parameter

The presence of the Direct Access ID self-defining parameter indicates that the auxiliary device can be accessed directly by using the destination/origin structured field. When multiple auxiliary devices are supported that use a product-defined data stream, separate Product-Defined Data Stream query replies must be provided, each of which has a unique DOID.

Table 139. IBM Product-Defined Direct Access Self-Defining Parameter			
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.

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 140. IBM DIA Base Format

Offset	Length	Content	Meaning
0	1 word	Length	Length of structure (includes self-defining parameters)
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 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 141.

Direct Access Self-Defining Parameter

The presence of the direct access ID self-defining parameter indicates that the auxiliary device can be accessed directly by using the destination/origin structured field.

Table 141. IBM Product-Defined Direct Access Self-Defining Parameter

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.

Appendix B. Compatibility with Personal Communications 4.1 for Windows 3.1

This appendix describes the compatibility between EHLLAPI/DDE of PC/3270 and PC400 for Windows 95 or Windows NT and that for Windows 3.1.

The EHLLAPI of PC/3270 and PC400 for Windows 95 or Windows NT is different from that for Windows 3.1 in the following ways:

The entry point name ***hllapiw*** that is provided by PC/3270 and PC400 for Windows 3.1 is not supported.

The event object is newly supported for the asynchronous mode of the notification used by **Start Close Intercept** (41), **Start Host Notification** (23), **Read Structured Fields** (126) and **Write Structured Fields** (127). It can be used by 32-bit applications for Windows 95 or Windows NT.

The DDE of PC/3270 and PC400 for Windows 95 or Windows NT is different from that for Windows 3.1 in the following ways:

Code conversion is not supported, use an appropriate API on Personal Communications for EBCDIC/ASCII conversion.

The application name ***IBM3270*** or ***IBM5250*** is for 16-bit applications. ***IBM327032*** and ***IBM525032*** are for 32-bit applications.

© Copyright IBM Corp. 1989, 1997

Appendix C. Compatibility with Communications Manager/2

This appendix describes the compatibility between EHLLAPI of PC/3270 and PC400 for Windows 95 and Windows NT and Communications Manager/2 (CM/2).

The following EHLLAPI function is not supported by PC/3270 or PC400:

Storage Manager (17)

The following EHLLAPI functions are different from those with the same names in CM/2. You need to understand the differences when you use these functions:

Set Session Parameter (9)
Copy OIA (13)
Copy String to PS (15)
Copy String to Field (33)
Get Key (51)
Window Status (104)
Query Sessions (10)
Connect for Structured Field (120)
Allocate Communications Buffer (123)
ASCII mnemonics

Set Session Parameter (9)

Set Options

Personal Communications does not provide the following set options provided by Communications Manager:

OLDOIA, NEWOIA
COMPCASE, COMPICASE
OLD5250OIA, NEW5250OIA

Return Parameters

When the **Set Session Parameter** (9) function is terminated, Communications Manager returns a length of the valid data string as the third parameter, the data string length. However, Personal Communications returns a number of the valid set options as the data string length.

EAB Option

In CM/2, a color remap affects the value of the character color in the EAB attribute copied by **Copy PS** (5) or **Copy PS to String** (8) function when the EAB option is specified in the **Set Session Parameter** (9) function.

In Personal Communications , however, the value of the character color in the EAB attribute depends on the contents of the presentation space regardless of a color remap, and it is not affected by a color remap.

© Copyright IBM Corp. 1989, 1997

Copy OIA (13)

The **Copy OIA** (13) function has the following differences between CM/2 and PC/3270 or PC400. For more information of the group and the column positions, refer to “Copy OIA (13)” on page 47.

Byte Position 21

- Personal Communications returns X'F6'.
- CM/2 returns X'20'.

Byte Positions 61–63

- Personal Communications does not return the printer information.
- CM/2 returns the printer information.

Group 3: Shift State

CM/2 does not return the value of bit 2. Bit 2 is reserved, and bit 0 contains both the Upper Shift and the Caps Lock.

Group 8 Byte 1: Input Inhibited

- Personal Communications does not return bit 6 (Device not working).
- CM/2 can return bit 6.

Group 8 Byte 3: Input Inhibited

- Personal Communications does not return bit 1 (Operator unauthorized) and bit 2 (Operator unauthorized -f).
- CM/2 can return bits 1 and 2.

Group 8 Byte 4: Input Inhibited

- Personal Communications does not return bit 2 (System wait).
- CM/2 can return bit 2.

Group 10: Highlight Group 2

- Personal Communications does not return bit 0 (Selected).
- CM/2 can return bit 0.

Group 11: Color Group 2

- Personal Communications does not return bit 0 (Selected).
- CM/2 can return bit 0.

Group 13: Printer Status

- In Personal Communications , this group is reserved.
- CM/2 can return this group.

Group 14: Graphics

CM/2 does not return bit 0 (Graphic cursor).

Copy String to PS (15)

In CM/2, the EAB option of the **Set Session Parameter** (9) function affects the **Copy String to PS** function. When you specify the EAB option, pass the attribute data that has the same size as the text data to the function with the text data.

In Personal Communications , however, the data to be passed is only text data regardless of EAB option. If you want to use the same interface with CM/2, use the PUTEAB option of **Set Session Parameter** (9).

Storage Manager (17)

Storage Manager (17) function provided by CM/2 is not supported by Personal Communications . Use the APIs provided by Windows 95 and Windows NT to allocate the memory for the applications.

Copy String to Field (33)

In CM/2, when the EAB option of the **Set Session Parameter** (9) function is specified, the attribute data is passed to the function as a part of the data. Therefore, when you specify the EAB option, pass the attribute data that has the same size as the text data to the function with the text data.

In Personal Communications , however, the EAB option does not affect the data contents of the **Copy String to Field** (33) function. The data to be passed is not the attribute data, but only the text data. If you want to use the same interface with CM/2, use the PUTEAB option of **Set Session Parameter** (9).

Get Key (51)

CM/2 returns shift state using @A, @S, or @r, if the shift state of a passed key is not a key or function recognized by the emulator session. Personal Communications does not support these ASCII mnemonics.

Window Status (104)

EHLLAPI function 104 (PM_WINDOW_STATUS) 'query extended status' command (0x03) will return the handle of the emulator presentation space window. This is consistent with the definition of the function and the CM/2 implementation. However, Personal Communications AS/400 and 3270 for Windows EHLLAPI return the handle of the frame window. EHLLAPI applications written for Personal Communications AS/400 and 3270 for Windows using this function need to use the parent of the window handle returned.

Query Sessions (10)

In CM/2 and Personal Communications for OS/2, the descriptor for personal computer is returned.

But in Personal Communications for Window, it is not returned.

Connect for Structured Fields (120)

The event object for communication connection status provided by CM/2 is not in Personal Communications .

Allocate Communications Buffer (123)

In CM/2, the maximum value of the requested buffer size is 64 KB minus 8 bytes (X'FFF8').

In Personal Communications , however, it is 64 KB minus 256 bytes (X'FF00').

ASCII Mnemonics

The following ASCII mnemonics are not supported in Personal Communications :

Mnemonics	Meaning
@A@N	Get Cursor
@A@O	Locate Cursor
@A@X	Hexadecimal
@A@Y	Cmd (Function) Key
@A@a	Destructive Backspace
@S@A	Erase EOL
@S@B	Field Advance
@S@C	Field Backspace
@S@D	Valid Character Backspace
@S@P	POR (For sending only)
@S@T	Jump to Task Manager
@/	Overrun of queue (Only in the **Get Key** function)

Appendix D. DOS-Mode EHLLAPI

IBM eNetwork Personal Communications Version 4. 2 supports EHLLAPI applications for DOS. This appendix provides information about this support.

Installation

To install DOS EHLLAPI support for IBM eNetwork Personal Communications Version 4. 2 do the following:

1. Select the Emulator Utilities folder from the Utilities folder in the IBM Personal Communications folder.
2. Select DOS EHLLAPI application from the Emulator Utilities folder.
3. Select the check box of the DOS MODE EHLLAPI to enable DOS EHLLAPI support.
4. Enter the major DOS version for which your DOS EHLLAPI applications are written. (For example, 2 for DOS Emulator Version 2.x).
5. Select OK to enable changes.
6. Shut down the workstation and restart it again.

For windows NT, this procedure adds the following statement in config.nt.

device=%SystemRoot%\system32\drivers\hlldrvr.sys

For Windows 95, this procedure adds the following statement in system.ini.

device=<drive:>\windows\system\doshll.vxd

Note: DOS EHLLAPI applications assert interrupt X'7F' to request EHLLAPI services. Any other proprietary DOS application using interrupt X'7F' will not work with DOS EHLLAPI enabled and vice-versa.

Compatibility

An EHLLAPI of IBM eNetwork Personal Communications Version 4. 2 is different from the one for windows in the following ways:

The Asynchronous Update Notification of **Start Close Intercept (41)** function is not supported.

The Asynchronous Update Notification of **Start Host Notification (23)** function is not supported.

The entry point name **hllapiw**, is not supported.

The EHLLAPI of IBM eNetwork Personal Communications Version 4. 2 is different from the one for DOS in the following ways.

Storage Manager (27) function is not supported.

© Copyright IBM Corp. 1989, 1997

Appendix E. SRPI Return Codes

This appendix describes error handling in the SRPI environment. Types 0, 1, 2, and 3 return codes and their definitions are listed. Exception class definitions, code values, and object values are listed. Server return codes are also discussed.

Error Handling

An unsuccessful service request in the SRPI environment can result from problems at any of the different layers. SRPI shields applications from transport layer errors as much as possible. Errors within server processing are handled by the applications. The other errors are caused by SRPI and are treated accordingly.

Transport Layer Errors

SRPI tries to recover from transport layer errors. When recovery is not possible, SRPI returns to the requester with a return code indicating transport layer failure. The programmer should handle such failures using the problem determination procedures of the transport mechanism.

Application Errors

SRPI is responsible for routing requests to servers and returning replies to requesters. Requesters and servers are responsible for handling errors (except for abend) that servers encounter. When a server ends abnormally, SRPI returns to the requester with an abend notice in the SRPI return code.

The server return code is set by the server on the IBM host computer running under VM or MVS. The value and meaning of the server return code is dependent on the requester or the server.

SEND_REQUEST Processing Errors

SRPI return codes can encounter a number of errors in processing the **SEND_REQUEST** function. Such errors include:

Incorrect function parameters
Unidentified server
Inability to contact the server

There are also system error codes for internal SRPI errors.

Types of SRPI Return Codes

SRPI return codes include types 0, 1, 2, and 3:

Type 0 Indicates successful completion of the **SEND_REQUEST** function.

Type 1 Indicates errors detected by the SRPI router that prevent a request from being processed.

Type 2 Indicates errors detected by the SRPI router and reported to the remote computer by an acknowledge interchange unit.

Type 3 Indicates errors detected by the remote computer and reported to the SRPI router by an acknowledge interchange unit.

© Copyright IBM Corp. 1989, 1997

The return code values are word-reversed and byte-reversed within each word. For example, the SRPI return code X'0100 0402' is stored in the CPRB memory as X'0204 0001'.

Type 0 Return Code Definitions

The type 0 return code (constant return code UERERROK) has the following format: X'0000 0000'. This return code value indicates that the SRPI function completed successfully.

Type 1 Return Code Definitions

Type 1 return codes have the following format: X'0100 nnnn'.

The *nnnn* bytes are the hexadecimal value that indicates the specific error detected.

The return code definitions and descriptions are listed in Table 142.

Table 142 (Page 1 of 3). Type 1 Return Code Definitions and Descriptions

Hexadecimal Return Code	Constant Return Code	Description
X'0100 0402'	UERERRT1START	SRPI is not started because the host ECF program is not started.
X'0100 0404'	UERERRT1LOAD	The SRPI router is not loaded.
X'0100 0408'	UERERRT1BUSY	The SRPI router is busy. This return code is not used by the Personal Communications program.
X'0100 040A'	UERERRT1VER	The version ID in the CPRB passed to the SRPI router is not supported by the resident portion of the SRPI router. The version ID is automatically put into the CPRB by the C interface facility.
X'0100 040C'	UERERRT1EMU	Personal Communications is not loaded.
X'0100 040E'	UERERRT1ROUT	The server name supplied in the CPRB is not defined in the server routing table. Default routing is not configured so SRPI is unable to route the request. Use a valid server name or update the configuration to include the server name.
X'0100 0410'	UERERRT1COMMR	Communications resource not available.
X'0100 0412'	UERERRT1REST	3270 emulation has been restarted since the application last used SRPI. End the application and restart it before using SRPI.
X'0100 0414'	UERERRT1INUSE	The request has been routed to a communication session that is in use by File Transfer.
X'0100 0602'	UERERRT1QPLEN	Request parameters length exceeds the maximum value. The maximum value allowed is 32763.
X'0100 0604'	UERERRT1RPLEN	Reply parameters buffer length exceeds the maximum value. The maximum value allowed is 32763.

Table 142 (Page 2 of 3). Type 1 Return Code Definitions and Descriptions

Hexadecimal Return Code	Constant Return Code	Description
X'0100 0606'	UERERRT1VERB	Incorrect or unsupported verb type. The verb type in the CPRB passed to the SRPI router is not recognized. The verb type is put into the CPRB automatically by the C interface facility.
X'0100 0608'	UERERRT1SERV	Incorrect server name. One or more characters in the server name could not be converted to EBCDIC for sending to the host.
X'0100 060C'	UERERRT1QPAD	One of the following conditions exists: The request parameter address is not valid. The request parameter length extends beyond the end of the request parameter buffer. The request parameter address is 0 with a nonzero request parameter length.
X'0100 060E'	UERERRT1QDAD	One of the following conditions exists: The request data address is not valid. The request data length extends beyond the end of the request data buffer. The request data address is 0 with a nonzero request data length.
X'0100 0610'	UERERRT1RPAD	One of the following conditions exists: The reply parameter buffer address is not valid. The reply parameter buffer length extends beyond the end of the reply parameter buffer. The reply parameter buffer address is 0 with a nonzero reply parameter length.
X'0100 0612'	UERERRT1RDAD	One of the following conditions exists: The reply data buffer address is not valid. The reply data buffer length extends beyond the end of the reply data buffer. The reply data buffer address is 0 with a nonzero reply data length.
X'0100 0616'	UERERRT1TOPV	The TopView environment is not supported. This return code is not used by the Personal Communications program.
X'0100 0622'	UERERRT1INV3270 d	Notification of 3270 screen update indicator is not valid. The Notification of 3270 screen update indicator must be set to X'00' (notify user of 3270 screen update) or X'FF' (suppress user notification of 3270 screen update) in the CPRB.

Table 142 (Page 3 of 3). Type 1 Return Code Definitions and Descriptions		
Hexadecimal Return Code	**Constant Return Code**	**Description**
X'0100 0624'	UERERRT1INVCPRB	Incorrect CPRB segment. The CPRB address points to a truncated CPRB structure. Use a read/write data segment large enough to contain the entire CPRB structure.
X'0100 0802'	UERERRT1CNCL	The remote computer canceled the communication session while the request was being processed. You can cause this to happen by stopping the remote program with the F3 key in the emulator session. However, use of this value is not limited to user-initiated cancellation of the session. It is used any time SRPI receives notification from the host that the session is canceled while processing a request.
X'0100 0C00'	UERERRT1CONV	A system error occurred. Conversation with the host ended for one of the following reasons: The host communication session is not active. A link-level communication error occurred. The system was unable to transmit data reliably to or from the host. For example, a sequence error occurred.
X'0100 0C02'	UERERRT1ISE	A system error occurred because of an internal software error in the SRPI router.
X'0100 0C04'	UERERRT1PROT	A system error occurred. This is a protocol violation error or a system software error in the SRPI router or the host.
X'0100 0C06'	UERERRT1SYIN	A system error occurred. The error is caused by system inconsistency. This is a system software error in the SRPI router.

Type 2 Return Code Definitions

Type 2 return codes have the following format: X'02xx yyzz'.

The 3 error-specific bytes consist of the following exception conditions from the acknowledge interchange unit:

xx exception class
yy exception code
zz exception object

Note: No constants are supplied.

Type 3 Return Code Definitions

Type 3 return codes have the following format: X'03xx yyzz'.

The 3 error-specific bytes consist of the following exception conditions from the acknowledge interchange unit:

xx exception class
yy exception code

zz exception object

The return code definitions and descriptions are listed in Table 143.

Table 143. Type 3 Return Code Definitions and Descriptions

Hexadecimal Return Code	Constant Return Code	Description
X'0304 1D00'	UERERRT3NORES	A resource required by the host SRPI router to process the request is not available. This might be a temporary condition.
X'0304 1E00'	UERERRT3NOSER	The server is unknown at the host.
X'0304 1F00'	UERERRT3UNSER	The server is not available at the host.
X'0304 2200'	UERERRT3TERMS	The server terminated in a normal fashion but did not send a reply.
X'0304 2300'	UERERRT3ABNDS	The server terminated abnormally and did not send a reply.

Class Definitions for Type 2 and Type 3

The exception classes are syntax, semantic, and process.

Syntax exception class. This class reports violations of the transmission unit syntax rules (for example, omitting the server return code parameter: X'0202 1A08'). In general, a return code reporting a syntax exception indicates a system software error in the SRPI router or in the host.

Semantic exception class. This class reports conflicting parameters (for example, an incorrect correlation value: X'0203 1B00'). In general, a return code reporting a semantic exception indicates a system software error in the SRPI router or in the host.

Process exception class. This class reports exception conditions during request processing (for example, server unknown: X'0304 1E00').

The exception class definitions are listed in Table 144.

Table 144. Class Definitions for Type 2 and Type 3

Value	Definition
X'00' to X'01'	Reserved
X'02'	Syntax
X'03'	Semantic
X'04'	Process
X'05' to X'FF'	Reserved

Exception Code Values for Type 2 and Type 3

The exception code defines a specific error condition and is required with every error. The exception code values are listed in Table 145.

Table 145. Exception Code Values for Type 2 and Type 3	
Value	**Definition**
X'00'	Reserved
X'08'	Segmentation
X'0C'	Incorrect operand ID
X'0F'	Incorrect length
X'16'	Incorrect subfield type
X'18'	Incorrect subfield value
X'19'	Required operand missing
X'1A'	Required subfield missing
X'1B'	Correlation error
X'1C'	Data exceeds allowable maximum length
X'1D'	Resource not available
X'1E'	Server unknown
X'1F'	Server not available
X'20'	Parameter length
X'21'	Data length
X'22'	Normal termination
X'23'	Abnormal termination (server abend)
X'24'	Multiple occurrences of a subfield
X'25'	Multiple occurrences of operand

Note: All exception code values not specified in this table are reserved.

Exception Object Values for Type 2 and Type 3

The exception object defines the incorrect transmission unit object. An exception object is required with syntax errors. The exception object values are listed in Table 146.

Table 146. Exception Object Values for Type 2 and Type 3	
Value	**Definition**
X'00'	Not specified
X'01'	Prefix
X'07'	Command operand
X'08'	Command subfields
X'1C'	Parameters operand
X'1D'	Data operand
X'13'	Suffix

Note: All exception object values not specified in this table are reserved.

Server Return Codes

A server return code is a doubleword (4-byte) return code supplied by the server program and is returned to the requester program. The contents and meaning of the return status are defined by the requester or the server. For information about server return codes, contact your host personnel or see one of the following manuals:

TSO/E Version 2 Guide to the Server-Requester Programming Interface

IBM Programmer's Guide to the Server-Requester Programming Interface for VM/System Product

Appendix F. DDE Functions in a 16-Bit Environment

This appendix describes DDE functions in 16-bit mode. This is useful information when you are migrating from 16-bit to 32-bit mode.

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 6, “Using DDE Functions with a DDE Client Application” on page 251.

PC/3270 and PC400 DDE Data Items in a 16-Bit Environment

Microsoft Windows DDE uses a three-level naming scheme to identify data items: application, topic, and item. Table 147 on page 319 describes these levels.

Table 147. 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 PC400support one application name as Windows DDE server.

Table 148. Applications for PC/3270 and PC400

Application
IBM3270 IBM5250

The following table lists the topics you can use.

© Copyright IBM Corp. 1989, 1997

Table 149. 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
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 in a 16-Bit Environment

The following table lists the DDE functions that are available for use with PC/3270 or PC400 and the page

Function	PC/3270 Windows	PC400	Page
Find Field	Yes	Yes	322
Get Keystrokes	Yes	Yes	324
Get Mouse Input	Yes	Yes	325
Get Number of Close Requests	Yes	Yes	328
Get Operator Information Area	Yes	Yes	329
Get Partial Presentation Space	Yes	Yes	329
Get Presentation Space	Yes	Yes	332
Get Session Status	Yes	Yes	334
Get System Configuration	Yes	Yes	336
Get System Formats	Yes	Yes	337
Get System Status	Yes	Yes	338
Get System SysItems	Yes	Yes	339
Get System Topics	Yes	Yes	340
Get Trim Rectangle	Yes	Yes	341
Initiate Session Conversation	Yes	Yes	341
Initiate Structured Field Conversation	Yes	No.	342
Initiate System Conversation	Yes	Yes	343
Put Data to Presentation Space	Yes	Yes	343
Search for String	Yes	Yes	344
Send Keystrokes	Yes	Yes	346
Session Execute Macro	Yes	Yes	347
Set Cursor Position	Yes	Yes	353
Set Mouse Intercept Condition	Yes	Yes	354
Set Presentation Space Service Condition	Yes	Yes	356
Set Session Advise Condition	Yes	Yes	358
Set Structured Field Service Condition	Yes	No	359
Start Close Intercept	Yes	Yes	360
Start Keystroke Intercept	Yes	Yes	361
Start Mouse Input Intercept	Yes	Yes	363
Start Read SF	Yes	No	366
Start Session Advise	Yes	Yes	367
Stop Close Intercept	Yes	Yes	369
Stop Keystroke Intercept	Yes	Yes	369
Stop Mouse Input Intercept	Yes	Yes	370

Function	PC/3270 Windows	PC400	Page
Stop Read SF	Yes	No	371
Stop Session Advise	Yes	Yes	371
Terminate Session Conversation	Yes	Yes	372
Terminate Structured Field Conversation	Yes	No	372
Terminate System Conversation	Yes	Yes	373
Write SF	Yes	No	373

Refer to “Summary of DDE Functions in a 16-Bit Envionment” on page 374 for a summary of the 16-bit 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

Find Field

3270	*5250*	*VT*
Yes	Yes	Yes

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. This 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
bb or T*b*	This field.
P*b*	The previous field, either protected or unprotected.
N*b*	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 *b* 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, one of the following status codes is 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

3270	*5250*	*VT*
Yes	Yes	Yes

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. This 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, one of the following status codes is 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    0 // Pure text, no command
WC_TOKEN          1 // including commands
```

Get Mouse Input

3270	*5250*	*VT*
Yes	Yes	Yes

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, one of the following status codes is 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	

Parameter Name	Meaning	Value
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

3270	*5250*	*VT*
Yes	Yes	Yes

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. This 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, one of the following status codes is 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

3270	*5250*	*VT*
Yes	Yes	Yes

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, one of the following status codes is 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
}    OIADATA, far *lpOIADATA;
```

Get Partial Presentation Space

3270	*5250*	*VT*
Yes	Yes	Yes

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, one of the following status codes is 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

3270	*5250*	*VT*
Yes	Yes	Yes

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, one of the following status codes is 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 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

3270	*5250*	*VT*
Yes	Yes	Yes

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, one of the following status codes is 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)
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.

Fields	Returned values	Description
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: 0: The presentation space is unlocked. 4: The presentation space is busy. 5: The presentation space is locked.
Session Window Handle	XXXX	Window handle of the session.

Notes:

1. The status of each field is updated each time the status is requested.
2. A new field might be added in a future version of PC/3270 or PC400.

Get System Configuration

3270	*5250*	*VT*
Yes	Yes	Yes

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
Reserved	XXXXXX	Reserved
Reserved	XXXX	Reserved
Monitor Type	MONO, CGA, EGA, VGA, XGA	Type of the monitor
Country Code	NNNN	Country code used with 3270 or 5250

Get System Formats

3270	*5250*	*VT*
Yes	Yes	Yes

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 is 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

3270	*5250*	*VT*
Yes	Yes	Yes

The **Get System Status** function returns the status of each 3270 or 5250

```
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 is 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

3270	*5250*	*VT*
Yes	Yes	Yes

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 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 is 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

3270	*5250*	*VT*
Yes	Yes	Yes

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 topics are supported by PC/3270 or PC400:

System – System Topic
SessionA – Session A Topic [2]
⋮
SessionZ – Session Z Topic [2]

If PC/3270 or PC400 cannot return the list of topics, a DDE ACK message is returned with an error code in the low-order byte of the wStatus word:

WM_DDE_ACK(wStatus, aTOPICS)

[2] The actual number of session topics supported depends 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.

Return Code	Explanation
9	A system error occurred.

Get Trim Rectangle

3270	*5250*	*VT*
Yes	Yes	Yes

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. This 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, one of the following status codes is 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

3270	*5250*	*VT*
Yes	Yes	Yes

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(aIBM327032, aSessionN) );
```

where:

aIBM327032
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(aIBM327032, 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(aIBM327032, aSystem)
WM_DDE_ACK(aIBM327032, aSessionA)
⋮
WM_DDE_ACK(aIBM327032, 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

3270	*5250*	*VT*
Yes	Yes	Yes

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(aIBM3270, 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 returns an acknowledgment message with the following parameter:

WM_DDE_ACK(aIBM3270, aLUN_xxxx)

Initiate System Conversation

3270	*5250*	*VT*
Yes	Yes	Yes

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(aIBM327032, aSystem) );
```

where:

aIBM327032
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(aIBM327032, aSystem)

Put Data to Presentation Space

3270	*5250*	*VT*
Yes	Yes	Yes

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     0 // 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 one of 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

3270	*5250*	*VT*
Yes	Yes	Yes

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. This 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, one of the following status codes is 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

3270	*5250*	*VT*
Yes	Yes	Yes

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     0 // 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 one of 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.

Return Code	Explanation
2	Type of key strokes is not valid.
6	The specified format is not valid.
9	A system error occurred.

Session Execute Macro

3270	*5250*	*VT*
Yes	Yes	Yes

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.

ote:

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:
For OS/400 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 4.2 Reference for Windows 95 and Windows NT* or *Personal Communications Version 4.2 Reference for Windows 95 and Windows NT* 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:
For OS/400 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 4.2 Reference for Windows 95 and Windows NT* or *Personal Communications Version 4.2 Reference for Windows 95 and Windows NT* 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:

1. Text strings are enclosed in double quotation marks.
   Macros are prefixed with “macro”.
   The argument string for SENDKEY must be 255 characters or less.
   The following commands are supported.

Table 150 (Page 1 of 4). SENDKEY Command List

Command Name	Token	PC/3270	PC400
Alternate Cursor	alt cursor	Yes	Yes
Alternate Viewing Mode	alt view	Yes	Yes
Attention	sys attn	Yes	Yes
Backspace	backspace	Yes	Yes
Back Tab	backtab	Yes	Yes
Backtab Word	backtab word	Yes	Yes
Character Advance	character advance	No	Yes
Character Backspace	backspace valid	No	Yes
Clear Screen	clear	Yes	Yes
Clicker	click	Yes	Yes

Table 150 (Page 2 of 4). SENDKEY Command List

Command Name	Token	PC/3270	PC400
Color Blue	blue	Yes	No
Color Field Inherit	field color	Yes	No
Color Green	green	Yes	No
Color Pink	pink	Yes	No
Color Red	red	Yes	No
Color Turquoise	turquoise	Yes	No
Color White	white	Yes	No
Color Yellow	yellow	Yes	No
Cursor Blink	cursor blink	Yes	Yes
Cursor Down	down	Yes	Yes
Cursor Left	left	Yes	Yes
Cursor Right	right	Yes	Yes
Cursor Select	cursor select	Yes	Yes
Cursor Up	up	Yes	Yes
Delete Character	delete char	Yes	Yes
Delete Word	delete word	Yes	Yes
Device Cancel	device cancel	Yes	Yes
Dup Field	dup	Yes	Yes
Edit Clear	edit-clear	Yes	Yes
Edit Copy	edit-copy	Yes	Yes
Edit Cut	edit-cut	Yes	Yes
Edit Paste	edit-paste	Yes	Yes
Edit Undo	edit-undo	Yes	Yes
End Field	end field	Yes	Yes
Enter	enter	Yes	Yes
Erase EOF	erase eof	Yes	Yes
Erase Field	erase field	Yes	No
Erase Input	erase input	Yes	Yes
Fast Cursor Down	fast down	Yes	Yes
Fast Cursor Left	fast left	Yes	Yes
Fast Cursor Right	fast right	Yes	Yes
Fast Cursor Up	fast up	Yes	Yes
Field Exit	field exit	No	Yes
Field Mark	field mark	Yes	Yes
Field +	field +	No	Yes
Field –	field -	No	Yes
Graphic Cursor	+cr	Yes	No
Help	help	Yes	Yes
Highlighting Field Inherit	field hilight	Yes	No
Highlighting Reverse	reverse	Yes	No
Highlighting Underscore	underscore	Yes	No
Home	home	Yes	Yes

Table 150 (Page 3 of 4). SENDKEY Command List

Command Name	Token	PC/3270	PC400
Host Print	host print	Yes	No
Input	input	Yes	Yes
Input nondisplay	input nd	Yes	Yes
Insert Toggle	insert	Yes	Yes
Lower case	to lower	Yes	No
Mark Down	mark down	Yes	Yes
Mark Left	mark left	Yes	Yes
Mark Right	mark right	Yes	Yes
Mark Up	mark up	Yes	Yes
Move Mark Down	move down	Yes	Yes
Move Mark Left	move left	Yes	Yes
Move Mark Right	move right	Yes	Yes
Move Mark Up	move up	Yes	Yes
New Line	newline	Yes	Yes
Next Page	page down	No	Yes
Pause 1 second	pause	Yes	Yes
Previous Page	page up	No	Yes
Print Screen	local copy	Yes	Yes
Program Attention Key 1	pa1	Yes	No
Program Attention Key 2	pa2	Yes	No
Program Attention Key 3	pa3	Yes	No
Program Function Key 1 ⋮ Program Function Key 24	pf1 ⋮ pf24	Yes ⋮ X	Yes ⋮ X
Quit	quit	Yes	Yes
Reset	reset	Yes	Yes
Response Time Monitor	rtm	Yes	No
Roll Down	roll down	No	Yes
Roll Up	roll up	No	Yes
Rubout	rubout	Yes	Yes
Rule	rule	Yes	Yes
SO/SI Display	so si	Yes	Yes
SO/SI Generate	so si generate	No	Yes
System Request	sys req	Yes	Yes
Tab Field	tab field	Yes	Yes
Tab Word	tab word	Yes	Yes
Test	test request	No	Yes
Unmark	unmark	Yes	Yes
Upper case	to upper	Yes	No
Upper/Lower Change	to other	Yes	No

Table 150 (Page 4 of 4). SENDKEY Command List			
Command Name	**Token**	**PC/3270**	**PC400**
Wait for bind	wait app	Yes	Yes
Wait for System	wait sys	Yes	Yes
Wait transition	wait trn	Yes	Yes
Wait while input inh.	wait inp inh	Yes	Yes
Window Relocation 1 ⋮ Window Relocation 8	view 1 ⋮ view 8	Yes ⋮ X	Yes ⋮ X

Examples:

1. To logon

   [SENDKEY("Logon")]

2. To get reader list

   [SENDKEY("RDRL", enter)]

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("10 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

3270	*5250*	*VT*
Yes	Yes	Yes

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 PC400returns a positive ACK message to the client application. Otherwise, a negative ACK message is returned with one of 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

3270	*5250*	*VT*
Yes	Yes	Yes

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 PC400returns 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

3270	*5250*	*VT*
Yes	Yes	Yes

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      0 // Search forward.
WC_SRCHBKWD      1 // Search backward.
```

The following values are valid at the uEOFflag field:

```
WC_UNEFFECTEOF 0 // 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 one of 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

3270	*5250*	*VT*
Yes	Yes	Yes

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 367 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 one of 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

3270	*5250*	*VT*
Yes	Yes	Yes

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 one of 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

3270	*5250*	*VT*
Yes	Yes	Yes

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 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, 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

3270	*5250*	*VT*
Yes	Yes	Yes

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 one of 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

3270	*5250*	*VT*
Yes	Yes	Yes

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 PC400receives 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 one of 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 PC400returns 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 PC400returns 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

3270	*5250*	*VT*
Yes	Yes	Yes

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 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, 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

3270	*5250*	*VT*
Yes	Yes	Yes

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 358.

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 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, 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

3270	*5250*	*VT*
Yes	Yes	Yes

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 one of 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

3270	*5250*	*VT*
Yes	Yes	Yes

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 one of 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

3270	*5250*	*VT*
Yes	Yes	Yes

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 PC400can perform the DDE_UNADVISE, PC/3270 or PC400returns an ACK message containing positive status information to the client:

WM_DDE_ACK(wStatus, aMOUSE)

If PC/3270 or PC400cannot perform the DDE_UNADVISE, PC/3270 or PC400returns an ACK message containing negative status information and one of 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

3270	*5250*	*VT*
Yes	Yes	Yes

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 one of 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

3270	*5250*	*VT*
Yes	Yes	Yes

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 one of 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

3270	*5250*	*VT*
Yes	Yes	Yes

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

3270	*5250*	*VT*
Yes	Yes	Yes

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

3270	*5250*	*VT*
Yes	Yes	Yes

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

3270	*5250*	*VT*
Yes	Yes	Yes

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 one of 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 in a 16-Bit Envionment

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 151 (Page 1 of 5). Summary of DDE Functions in a 16-Bit Environment

Function name	Client command
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

Table 151 (Page 2 of 5). Summary of DDE Functions in a 16-Bit Environment

Function name	Client command
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
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

Table 151 (Page 3 of 5). Summary of DDE Functions in a 16-Bit Environment	
Function name	**Client command**
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(aIBM327032, aSessionN));
	N = a session letter A through Z.
Initiate Structured Field Conversation	SendMessage(-1, WM_DDE_INITIATE, hClientWnd, MAKELONG(aIBM327032, aLUN_xxxx)
	N = a session letter A through Z. xxxx = a user defined string.
Initiate System Conversation	SendMessage(-1, WM_DDE_INITIATE, hClientWnd, MAKELONG(aIBM327032, 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
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

Table 151 (Page 4 of 5). Summary of DDE Functions in a 16-Bit Environment	
Function name	**Client command**
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
Stop Close Intercept	PostMessage(hServerWnd, WM_DDE_UNADVISE, hClientWnd, MAKELONG(NULL, aCLOSE));
Stop Keystroke Intercept	PostMessage(hServerWnd, WM_DDE_UNADVISE, hClientWnd, MAKELONG(NULL, aKEYS));

Table 151 (Page 5 of 5). Summary of DDE Functions in a 16-Bit Environment	
Function name	**Client command**
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 in a 16-Bit Environment

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 in a 16-Bit Environment

To add, delete, and change menu items, the following DDE conversation must take place between the session and DDE menu server application.

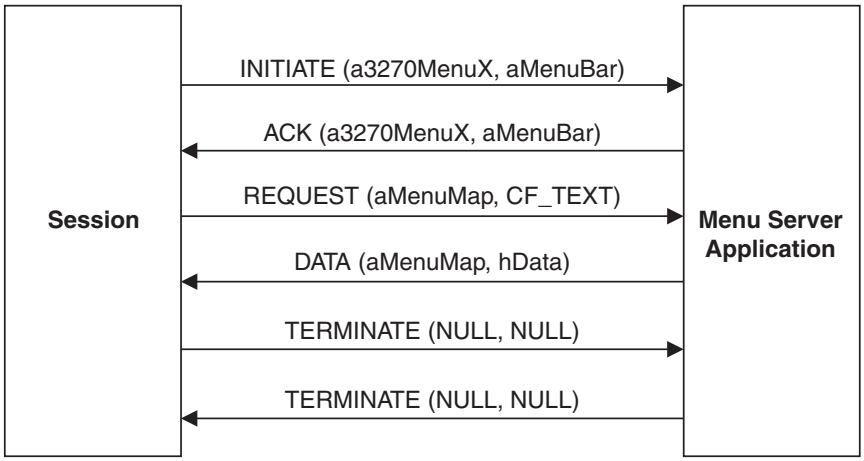


Figure 8. DDE Menu Server Conversation

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, 32-Bit

To add, delete, and change menu items, the following DDE conversation must take place between the session and a DDE menu client application.

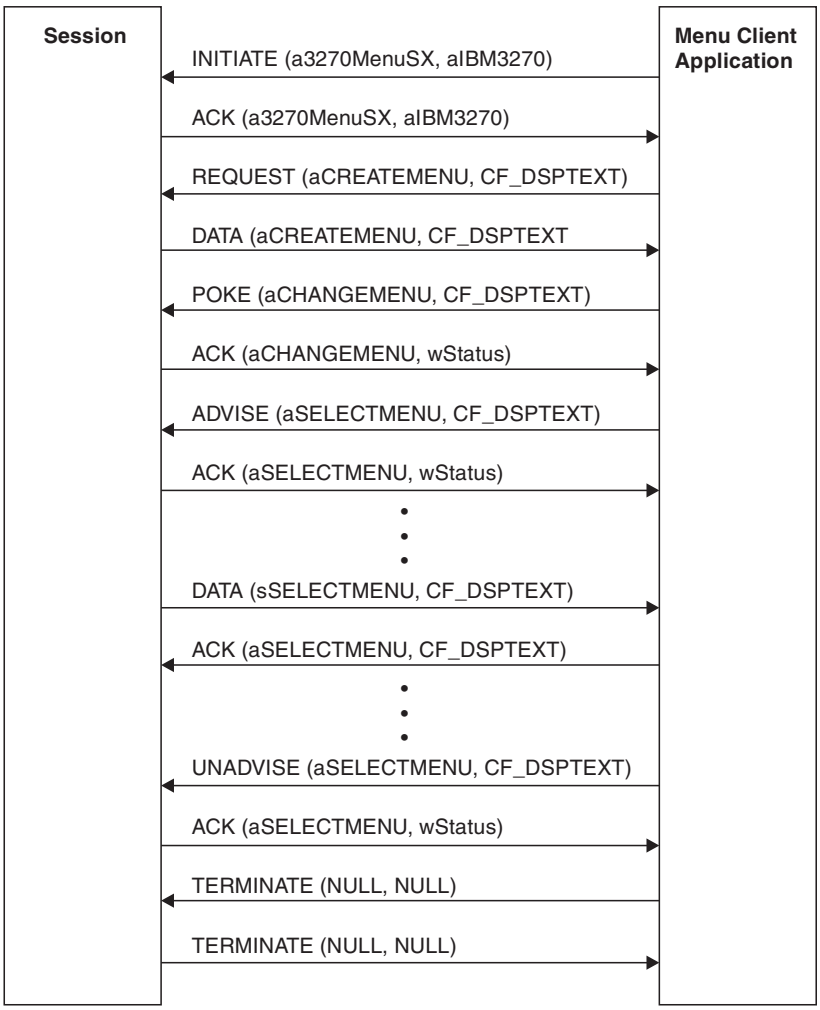


Figure 9. DDE Menu Client Conversation

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 in a 16-bit Environment

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	381
Create Menu Item	387
Initiate Menu Conversation	388
Start Menu Advise	389
Stop Menu Advise	390
Terminate Menu Conversation	391

Change Menu Item

3270	*5250*	*VT*
Yes	Yes	Yes

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 one of 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

3270	*5250*	*VT*
Yes	Yes	Yes

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, one of 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

3270	*5250*	*VT*
Yes	Yes	Yes

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(aIBM327032, SN) );
```

where:

aIBM327032

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.

SN

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(aIBM327032, SN)
```

Start Menu Advise

3270	*5250*	*VT*
Yes	Yes	Yes

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 0)
 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 one of 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

3270	*5250*	*VT*
Yes	Yes	Yes

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 one of 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

3270	*5250*	*VT*
Yes	Yes	Yes

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

Appendix G. REXX EHLLAPI Functions

This appendix assists application programmers who are using EHLLAPI to write REXX language application programs. You should be familiar with the REXX command language. An overview of REXX EHLLAPI functions is provided. The functions are listed in alphabetic order, and a detailed description accompanies each function. Information about writing applications using REXX and sample programs is included after the function descriptions.

REXX applications require the use of IBM Object REXX for Windows 95 and NT.

Overview of REXX EHLLAPI Function Calls and Return Values

REXX EHLLAPI is called by either a REXX function or a REXX subroutine. The REXX function returns a value, which is assigned to a variable or return code:

rc=**HLLAPI(** *function-string* [,*parameters*] **)**

The REXX subroutine places the return value in the special REXX variable called Result as follows:

call HLLAPI *function-string* [,*parameters*]

Installation

The REXX EHLLAPI executable file (SAAHLAPI.DLL) is installed for IBM Personal Communications.

EHLLAPI is provided as an external feature and is loaded into memory only when used. To ensure that REXX EHLLAPI functions are interpreted, REXX application programs must contain the following statement:

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

REXX EHLLAPI provides a *single* function name, which is used with a set of parameters. The function name is the first parameter of the rxfuncadd call above; the default is **HLLAPI**.

Conventions

Each REXX EHLLAPI function description contains the following:

- Function name
- Prerequisite calls
- Supplied syntax
- Supplied parameters
- Returned parameters
- Additional information

Function Name
Provides the name and a brief explanation of the function.

Prerequisite Calls
Lists any functions that your application program must call before you can use the present function. The word *None* indicates that no prerequisite calls are

© Copyright IBM Corp. 1989, 1997

required. Prerequisite calls for all REXX EHLLAPI functions are listed under “Summary of Prerequisite Calls for Functions” on page 394.

Supplied Syntax
Illustrates the syntax of the function call. Refer to “Overview of REXX EHLLAPI Function Calls and Return Values” on page 393 for more information.

Supplied Parameters
Lists the parameters that your program must define in order to call the discussed REXX EHLLAPI function.

Every function call uses the name of the function as the first parameter, but the number of parameters that your program must define depends on the function being called. Refer to each function description for the format of the supplied parameters.

Returned Parameters
Places values, as either a return code or actual data, in a single variable. If you are using REXX EHLLAPI as a function, these values are put in the receiving variable. If you are using REXX EHLLAPI as a subroutine, the values are placed in a special variable called *result*. Refer to each function description for the format of the returned values.

Additional Information
Provides any necessary technical information about the function.

Summary of Prerequisite Calls for Functions

Table 152 lists the prerequisite calls for each REXX EHLLAPI function. Prerequisite calls are required when you use an application program.

Table 152 (Page 1 of 2). Prerequisite Calls for Functions

Function	Prerequisite Calls
Change_Switch_Name	Connect_PM
Change_Window_Name	Connect_PM
Connect	None
Connect_PM	None
Convert_Pos	None
Copy_Field_To_String	Connect
Copy_OIA	Connect
Copy_PS	Connect
Copy_PS_To_Str	Connect
Copy_Str_To_Field	Connect
Copy_Str_To_PS	Connect
Disconnect	Connect
Disconnect_PM	Connect_PM
Find_Field_Len	Connect
Find_Field_Pos	Connect
Get_Key	Start_Keystroke_Intercept
Get_Window_Status	Connect_PM
Intercept_Status	Start_Keystroke_Intercept

Table 152 (Page 2 of 2). Prerequisite Calls for Functions	
Function	**Prerequisite Calls**
Lock_PMSVC	Connect_PM
Lock_PS	Connect
Pause	None
Query_Close_Intercept	Start_Close_Intercept
Query_Cursor_Pos	Connect
Query_Field_Attr	Connect
Query_Host_Update	Start_Host_Notify
Query_Session_Status	None
Query_Sessions	None
Query_System	None
Query_Window_Coord	Connect_PM
Receive_File	None
Release	Connect
Reserve	Connect
Reset_System	None
Search_Field	Connect
Search_PS	Connect
Send_File	None
Sendkey	Connect
Set_Cursor_Pos	Connect
Set_Session_Parms	None
Set_Window_Status	Connect_PM
Start_Close_Intercept	None
Start_Host_Notify	None
Start_Keystroke_Intercept	None
Stop_Close_Intercept	Start_Close_Intercept
Stop_Host_Notify	Start_Host_Notify
Stop_Keystroke_Intercept	Start_Keystroke_Intercept
Wait	Connect

Summary of EHLLAPI and REXX EHLLAPI Functions

Table 153 lists each EHLLAPI and REXX EHLLAPI function.

Table 153 (Page 1 of 2). EHLLAPI and REXX EHLLAPI Functions	
EHLLAPI	**REXX EHLLAPI**
CHANGE SWITCH LIST LT NAME (105)	Change_Switch__Name
CHANGE PS WINDOW NAME (106)	Change_Window_Name
CONNECT PRESENTATION SPACE (1)	Connect
CONNECT PM WINDOW SERVICES (101)	Connect_PM
CONVERT POSITION or CONVERT ROWCOL (99)	Convert_Pos
COPY FIELD TO STRING (34)	Copy_Field_To_String
COPY OIA (13)	Copy_OIA

Table 153 (Page 2 of 2). EHLLAPI and REXX EHLLAPI Functions

EHLLAPI	REXX EHLLAPI
COPY PRESENTATION SPACE (5)	Copy_PS
COPY PRESENTATION SPACE TO STRING (8)	Copy_PS_To_Str
COPY STRING TO FIELD (33)	Copy_Str_To_Field
COPY STRING TO PRESENTATION SPACE (15)	Copy_Str_To_PS
DISCONNECT PRESENTATION SPACE (2)	Disconnect
DISCONNECT PM WINDOW SERVICES (102)	Disconnect_PM
FIND FIELD LENGTH (32)	Find_Field_Len
FIND FIELD POSITION (31)	Find_Field_Pos
GET KEY (51)	Get_Key
PAUSE (18)	Pause
PM WINDOW STATUS (104)	Get_Window_Status and Set_Window_Status
POST INTERCEPT STATUS (52)	Intercept_Status
LOCK PMSVC API (61)	Lock_PMSVC
LOCK PRESENTATION SPACE API (60)	Lock_PS
QUERY CLOSE INTERCEPT (42)	Query_Close_Intercept
QUERY CURSOR LOCATION (7)	Query_Cursor_Pos
QUERY FIELD ATTRIBUTE (14)	Query_Field_Attr
QUERY HOST UPDATE (24)	Query_Host_Update
QUERY PM WINDOW COORDINATES (103)	Query_Window_Coord
QUERY SESSION STATUS (22)	Query_Session_Status
QUERY SESSIONS (10)	Query_Sessions
QUERY SYSTEM (20)	Query_System
RECEIVE FILE (91)	Receive_File
RELEASE (12)	Release
RESERVE (11)	Reserve
RESET SYSTEM (21)	Reset_System
SEARCH FIELD (30)	Search_Field
SEARCH PRESENTATION SPACE (6)	Search_PS
SEND FILE (90)	Send_File
SEND KEY (3)	Sendkey
SET CURSOR (40)	Set_Cursor_Pos
SET SESSION PARAMETERS (9)	Set_Session_Parms
START CLOSE INTERCEPT (41)	Start_Close_Intercept
START HOST NOTIFICATION (23)	Start_Host_Notify
START KEYSTROKE INTERCEPT (50)	Start_Keystroke_Intercept
STOP CLOSE INTERCEPT (43)	Stop_Close_Intercept
STOP HOST NOTIFICATION (25)	Stop_Host_Notify
STOP KEYSTROKE INTERCEPT (53)	Stop_Keystroke_Intercept
WAIT (4)	Wait

Change_Switch_Name

The **Change_Switch_Name** function changes or resets the name of the session listed on the window title bar.

Prerequisite Calls

Connect_PM

Supplied Syntax

The syntax for **Change_Switch_Name** is as follows:

HLLAPI('Change_Switch_Name', session_id, *type* [, *new_name*])

Supplied Parameters

Function name: **Change_Switch_Name**

session_id: The single-character short name of the session to be renamed on the Task List. If necessary, use the Communications Manager configuration menus or the **Query_Sessions** function to determine the short names of the sessions that are configured on the system.

*type***:** One of the following:

Set renames the session name (title) for session *session_id* to *new_name.*

Reset restores the original session name (title).

Note: Only the first character of **Set** or **Reset** is significant.

Returned Parameters

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

Change_Window_Name

The **Change_Window_Name** function changes or resets the name of the session listed on the host window title bar.

Prerequisite Calls

Connect_PM

Supplied Syntax

The syntax for the **Change_Window_Name** function is as follows:

HLLAPI('Change_window_name', *session_id*, *type* [, *new_name*])

Supplied Parameters

Function name: **Change_Window_Name**.

session_id: The single-character short name of the session to be renamed on the title bar.

type: One of the following:

Set changes the window name (title) for session **session_ id** to *new_name*.

Reset restores the original window name (title).

Note: Only the first character of **Set** or **Reset** is significant.

Returned Parameters

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

Connect

The **Connect** function connects the REXX application program to the host presentation space.

Prerequisite Calls

None.

Supplied Syntax

The syntax for the **Connect** function is as follows:

HLLAPI('Connect',session_id)

Supplied Parameters

Function name: **Connect**

session_id: The single-character short name of the session window you want to connect to. If necessary, use Communications Manager configuration menus or the **Query_Sessions** function to determine the names of the sessions configured on your system.

Returned Parameters

Return Code	Explanation
0	The **Connect** function was successful.
1	An incorrect host presentation space ID was specified. The specified session either does not exist or is a logical printer session. This return code could also mean that the API Setting for DDE/EHLLAPI is not set on.
4	Successful connection was achieved, but the session is busy.
5	Successful connection was achieved, but the session is locked (input-inhibited).
9	A system error occurred.
11	The session is already being used by another system function.

Connect_PM

The **Connect_PM** function connects the REXX application program to the presentation space window.

Prerequisite Calls

None.

Supplied Syntax

The syntax for the **Connect_PM** function is as follows:

HLLAPI('Connect_PM', *session_id* **)**

Supplied Parameters

Function name: **Connect_PM**

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

Returned Parameters

Return Code	Explanation
0	The **Connect_PM** function was successful.
1	An incorrect host presentation space ID was specified. This return code could also mean that the API Setting for DDE/EHLLAPI is not set on.
9	A system error occurred.
10	The function is not supported by your emulation program.
11	The session is already being used by another system function.

Convert_Pos

The **Convert_Pos** function converts the host presentation space positional value into the display row-column coordinates or converts the display row-column coordinates into the host presentation space positional value for the given session_ID.

Note: If row-column conversion is specified, *column* is the second parameter. The valid row-column values are the values you specified when you configured this session. For example, a session with 24 rows and 80 columns contains positions 1 to 1920.

Prerequisite Calls

None.

Supplied Syntax

The syntax for the **Convert_Pos** function is as follows:

HLLAPI('Convert_pos', *session_id*, *column* | *position* [, *row*] **)**

Supplied Parameters

Function name:	**Convert_Pos**
session_id:	The single-character short name of the session.
column:	The column to be converted. It must be specified with the row.
row:	The row to be converted.
position:	The position to be converted.

Returned Parameters

The following values are valid if row-column conversion is requested:

Return Code	Explanation
0	The specified row or column is outside the presentation space.
n	The position of the specified row and column. For example: HLLAPI ('Convert_pos', 'a', 10, 2) = '170' converts column 10, row 2 to 170 in a 24x80 presentation space.

The following values are valid if position conversion is requested:

Return Code	Explanation
0	The specified position is outside the presentation space.
c r	Where **c** is the column number and **r** is the row number. For example: HLLAPI ('Convert_pos', 'a', 170) = '10 2' converts position 170 to column 10, row 2 in a 24x80 presentation space.

Copy_Field_To_Str

The **Copy_Field_To_Str** function transfers characters from a target field into a data string. Use the **Find_Field_Pos** and **Find_Field_Len** functions to determine the target and its length values.

Prerequisite Calls

Connect

Supplied Syntax

The syntax for the **Copy_Field_To_Str** function is as follows:

HLLAPI('Copy_field_to_str', *pos*, *length*)

Supplied Parameters

Function name:	**Copy_Field_To_Str**
pos:	The target field to be copied.
length:	The length, in bytes, of the target data string.

Returned Parameters

Return Code	Explanation
''	Null. No field data was found at *pos*, or a *pos* that is not valid was specified.
data	The contents of the returned data string are determined by the extended attribute bytes (EAB) value in the **Set_Session_Parms** function. If EAB is set off, only text from the presentation space is returned. If EAB is set on, 2 bytes are returned for each byte that is displayed. The first byte contains the EAB value, and the second byte contains the text data. Double-byte EAD can be returned by using the EAD option of the (9) function with the **COPY FIELD TO STRING** function. If the EAD is specified without specifying the EAB option, EAD is returned after each character. If the EAB option is specified, EAD is returned after EAB.

Copy_OIA

This function returns the contents of the operator information area (OIA) from the connected session.

Prerequisite Calls

Connect

Supplied Syntax

The syntax for the **Copy_OIA** function is as follows:

```
HLLAPI( 'Copy_OIA' )
```

Supplied Parameter

Function name: **Copy_OIA**.

Returned Parameters

Return Code	Explanation
''	Null. Not connected or other error.
data	The 104-byte copy of the OIA data. Refer to “Copy OIA (13)” on page 47.

Copy_PS

The **Copy_PS** function returns the entire contents of the presentation space for the currently connected session.

Prerequisite Calls

Connect.

Supplied Syntax

The syntax for the **Copy_PS** function is as follows:

```
HLLAPI( 'Copy_PS' )
```

Supplied Parameter

Function name: **Copy_PS**.

Returned Parameters

Return Code	Explanation
''	Null. Not connected or other error.
data	The contents of the returned data string are determined by the EAB value in the **Set_Session_Parms** function. If EAB is set off, only text from the presentation space is returned. If EAB is set on, 2 bytes are returned for each byte that is displayed. The first byte contains the EAB value, and the second byte contains the text data. These bytes are returned as a space character if the start position of the copy is the second byte of the double-byte character or if the end position is the first byte of the double-byte character.

Additional Information

COPY_PS results in two calls to EHLLAPI. The first is **QUERY_SESSIONS**, which is used to determine the size of the presentation space. The second call, **COPY_PS_TO_STRING**, is used in place of **COPY_PS** in order to prevent possible buffer overflow.

COPY_PS_TO_STRING is used because of the slim possibility that the size of the presentation space might change between the calls to **QUERY_SESSIONS** and **COPY_PS_TO_STRING**.

Note: Should the presentation space increase in size, only the number of bytes returned on the **QUERY_SESSIONS** are copied. If the size decreases, then the characters beyond the current size should be ignored.

Copy_PS_To_Str

The **Copy_PS_To_Str** function copies data from the currently connected session into a data string.

Prerequisite Calls

Connect

Supplied Syntax

The syntax for the **Copy_PS_To_Str** function is as follows:

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

Supplied Parameters

Function name: **Copy_PS_To_Str**.

*pos***:** The target field to be copied.

*length***:** The length, in bytes, of the target data string.

Note: Do not double the target field value if EAB is set on. REXX EHLLAPI does this for you automatically, if EAD is set on.

Returned Parameters

Return Code	Explanation
''	Null. Not connected or other error.
data	The contents of the returned data string are determined by the EAB value in the **Set_Session_Parms** function. If EAB is set off, only text from the presentation space is returned. If EAB is set on, 2 bytes are returned for each byte that is displayed. The first byte contains the EAB value, and the second byte contains the text data.

Copy_Str_To_Field

The **Copy_Str_To_Field** function copies a string of characters into a specified field at the target field location position of the connected session.

Prerequisite Calls

Connect

Supplied Syntax

The syntax for the **Copy_Str_To_Field** function is as follows:

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

Supplied Parameters

Function name: **Copy_Str_To_Field**.

string: The string containing the data to be transferred to the target field.

pos: The target field to be copied.

Returned Parameters

Return Code	Explanation
0	The **Copy_Str_To_Field** function was successful.
1	Your program is not currently connected to the host session.
2	Parameter error.
5	The target field was protected or inhibited, or incorrect data was sent to the target field (such as a field attribute).
6	Copy was completed, but data was truncated.
7	The *pos* parameter was not valid.
9	A system error occurred.
24	The screen has no fields (unformatted).

Copy_Str_To_PS

The **Copy_Str_To_PS** function copies a string of characters into the host presentation space specified by the *pos* . parameter.

Prerequisite Calls

Connect

Supplied Syntax

The syntax for the **Copy_Str_To_PS** function is as follows:

HLLAPI('Copy_str_to_PS', *string*, *pos* **)**

Supplied Parameters

Function name: **Copy_Str_To_PS**.

string: The string containing the data to be transferred to the presentation space.

pos: The presentation space to be copied.

Returned Parameters

Return Code	Explanation
0	The **Copy_Str_To_PS** function was successful.
1	Your program is not currently connected to the host session.
2	Parameter error.
5	The target field was protected or inhibited, or incorrect data was sent to the target field (such as a field attribute).
6	Copy was completed, but data was truncated.
7	The *pos* parameter was not valid.
9	A system error occurred.

Double-byte characters

Disconnect

The **Disconnect** function disconnects your application program from the currently connected session.

Prerequisite Calls

Connect

Supplied Syntax

The syntax for the **Disconnect** function is as follows:

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

Supplied Parameter

Function name: **Disconnect**.

Returned Parameters

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

Disconnect_PM

The **Disconnect_PM** function disconnects from the session window.

Prerequisite Calls

Connect_PM

Supplied Syntax

The syntax for the **Disconnect_PM** function is as follows:

HLLAPI('Disconnect_PM', *session_id* **)**

Supplied Parameters

Function name: **Disconnect_PM**.

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

Returned Parameters

Return Code	Explanation
0	The **Disconnect_PM** function was successful.
1	Your program is not currently connected for Window Services.
9	A system error occurred.

Find_Field_Len

Prerequisite Calls

The **Find_Field_Len** function returns the length of the target field along with the attributes specified by the *search_option* parameter.

Connect

Supplied Syntax

The syntax for the **Find**_Field_Len function is as follows:

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

Supplied Parameters

Function name: **Find_Field_Len**.

*search_option***:** See the following table.

*pos***:** The target field to be copied.

The following *search_option* values are valid:

Value	Explanation
'*bb*' or 'T*b*_'	Current field (the field where the cursor is located).
'N*b*'	Next field, either protected or unprotected.
'P*b*'	Previous field, either protected or unprotected.
'NP'	Next protected field.
'NU'	Next unprotected field.
'PP'	Previous protected field.
'PU'	Previous unprotected field.

Returned Parameters

Return Code	Explanation
0	The specified field was not found.
data	The length of the specified field.

Find_Field_Pos

Prerequisite Calls

The **Find_Field_Pos** function returns the location of the target field with the attributes specified by the *search_option* parameter.

Connect

Supplied Syntax

The syntax for the **Find_Field_Pos** function is as follows:

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

Supplied Parameters

Function name: **Find_Field_Pos**.

*search_option***:** See the following table.

*pos***:** The target field to be copied.

The following *search_option* values are valid:

Value	Explanation
'*bb*_' or 'T*b*_'	The current field (the field where the cursor is located).
'N*b*_'	The next field, either protected or unprotected.
'P*b*_'	The previous 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.

Returned Parameters

Return Code	Explanation
0	The specified field was not found.
data	The position of the specified field.

Get_Key

Prerequisite Calls

The **Get_Key** function allows your application program to intercept keystrokes from the specified *session_id*, or from the currently connected session if *session_id* is blank. The program waits until a keystroke becomes available.

Start_Keystroke_Intercept

Supplied Syntax

The syntax for the **Get_Key** function is as follows:

HLLAPI('Get_key', *session_id* **)**

Supplied Parameters

Function name: **Get_Key**.

session_id: The single-character short name of the session.

Returned Parameters

Return Code	Explanation
''	Null. Error or not connected to *session_id*.
data string	The contents of the data string depend on the keys pressed by the 3270 or 5250 session operator. Refer to “Keyboard Mnemonics” on page 128.

Note: The @ (escape) character is set by specifying ESC= in the **Set_Session_Parms** function.

Additional Information

If keystroke interception is active (through the **Start_Keystroke_Intercept** function), no keystrokes are sent to the connected session until you perform the following tasks:

1. Specify the **Get_Key** function to remove the keystroke from the intercept buffer.
2. Specify the **Intercept_Status** function to either accept or reject the keystroke. If you specify *Accept*, the keystroke is sent to the connected session by the **Sendkey** function. If you specify *Reject*, the keystroke is discarded.

Get_Window_Status

Prerequisite Calls

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

Connect_PM

Supplied Syntax

The syntax for the **Get_Window_Status** function is as follows:

HLLAPI('Get_window_**status',** *session_id* **)**

Supplied Parameters

Function name: **Get_Window_Status**.

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

Returned Parameters

Return Code	Explanation
''	Null. Not connected to the window. Refer to the **Connect_PM** function for more information.
0008	The window is visible.
0010	The window is invisible.
0080	The window is activated.
0100	The window is deactivated.
0400	The window is minimized.
0800	The window is maximized.
Note: If more than one of the above states is true, the return codes are added together. For example, if the window is visible (0008), deactivated (0100), and maximized (0800), the return code is 0908.	

Intercept_Status

Prerequisite Calls

The **Intercept_Status** function informs the session when a keystroke obtained through the **Get_Key** function was accepted or rejected.

Start_Keystroke_Intercept

Supplied Syntax

The syntax for the **Intercept_Status** function is as follows:

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

Supplied Parameters

Function name: **Intercept_Status**.

session_id: The single-character short name of the session.

status: See the following table:

Value	Explanation
'A'	Accept the keystroke.
'R'	Reject the keystroke. Signal with a beep.

Returned Parameters

Return Code	Explanation
0	The **Interrupt_Status** function was successful.
1	The presentation space was not valid.
8	No prior **Start_Keystroke_Intercept** function was active.
9	A system error occurred.

Lock_PMSVC

Prerequisite Calls

This function locks or unlocks the presentation space window.

Connect_PM

Supplied Syntax

The syntax for the **Lock_PMSVC** function is as follows:

HLLAPI('Lock_PMSVC', session_id, status, queue_option)

Supplied Parameters

Function name: **Lock_PMSVC**.

session id: The single-character short name of the session.

status: See the following table:

Value	Explanation
'L'	Lock the presentation space window.
'U'	Unlock the presentation space window.

queue_option: See the following table:

Value	Explanation
'R'	Return immediately.
'Q'	Queue if the presentation space window is already locked (for lock only).

Returned Parameters

Return Code: The following values are valid:

Return Code	Explanation
0	The **Lock_PMSVC** function was successful.
1	An incorrect host presentation space was specified or not connected.
2	An error was made in specifying parameters.
9	A system error occurred.
43	The API was already locked by another EHLLAPI application (on LOCK), or API not locked (on UNLOCK).

Lock_PS

Prerequisite Calls

The **Lock_PS** function locks or unlocks the presentation space.

Connect

Supplied Syntax

The syntax for the **Lock_PS** function is as follows:

```
HLLAPI( 'Lock_PS', session_id, status, queue_option )
```

Supplied Parameters

Function name: **Lock_PS**.

session id: The single-character short name of the session.

status: See the following table:

Value	Explanation
'L'	Lock the presentation space.
'U'	Unlock the presentation space.

queue_option: See the following table:

Value	Explanation
'R'	Return immediately.
'Q'	Queue if the presentation space is already locked (for LOCK only).

Returned Parameters

Return Code: The following values are valid:

Return Code	Explanation
0	The **Lock_PS** function was successful.
1	The presentation space was not valid.
2	An error was made in specifying parameters.
9	A system error occurred.
43	The API was already locked by another EHLLAPI application (on LOCK), or API not locked (on UNLOCK).

Pause

Prerequisite Calls

The **Pause** function causes a timed pause of *n* ½-second intervals to occur.

If the **Set_Session_Parms** function is set to IPAUSE and a **Start_Host_Notify** function has been called, the pause is also ended by an update to the host screen. If *sessname* is provided and IPAUSE has been set, only updates to the specified session interrupts the pause. Otherwise, updates to any connected session interrupts the pause (if IPAUSE has been set).

None.

Supplied Syntax

The syntax for the **Pause** function is as follows:

HLLAPI('Pause', *n* [, *sessname*] **)**

Supplied Parameters

Function name: **Pause**.

*n***:** A timed pause.

*sessname***:** An optional parameter. See the following table:

Value	Explanation
'X#'	X is the name of the short-session_id, and # is coded exactly as shown.

Returned Parameters

Return Code	Explanation
0	The wait duration has expired.
9	A system error occurred.
26	The host session presentation space or OIA has been updated. Refer to “Query_Host_Update” on page 421 for more information.

Query_Close_Intercept

Prerequisite Calls

The **Query_Close_Intercept** function determines if a close request was started from the session.

Start_Close_Intercept

Supplied Syntax

The syntax for the **Query_Close_Intercept** function is as follows:

HLLAPI('Query_close_**intercept',** *session_id* **)**

Supplied Parameters

Function name: **Query_Close_Intercept**.

*session_id***:** The single-character short name of the host session.

Returned Parameters

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

Query_Cursor_Pos

Prerequisite Calls

The **Query_Cursor_Pos** function returns the cursor position for the currently connected session.

Connect

Supplied Syntax

The syntax for the **Query_Cursor_Pos** function is as follows:

HLLAPI('Query_cursor_**pos')**

Supplied Parameter

Function name: **Query_Cursor_Pos**.

Returned Parameters

Return Code	Explanation
0	Your program is not currently connected to the host session.
data	Cursor position.

Query_Field_Attr

Prerequisite Calls

The **Query_Field_Attr** function returns the hexadecimal representation of the field attribute in the currently connected session.

Connect

Supplied Syntax

The syntax for the **Query_Field_Attr** function is as follows:

HLLAPI('Query_field_**attr',** *pos* **)**

Supplied Parameters

Function name: **Query_Field_Attr**.

*pos***:** The target field to be copied.

Returned Parameters

Return Code	Explanation
1	Your program is not currently connected to the host session.
data	Attribute bytes (printable hexadecimal characters equal to or greater than X'C0').

Query_Host_Update

Prerequisite Calls

The **Query_Host_Update** function determines if the OIA or presentation space for the session has been updated.

Start_Host_Notify

Supplied Syntax

The syntax for the **Query_Host_Update** function is as follows:

HLLAPI('Query_host_**update',** *session_id* **)**

Supplied Parameters

Function name: **Query_Host_Update**.

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

Returned Parameters

Return Code	Explanation
0	No updates were made since the last call.
1	An incorrect host presentation space was specified.
8	No prior **Start_Host_Notify** function was called for the host presentation space ID.
9	A system error occurred.
21	The OIA has been updated.
22	The presentation space was updated.
23	The OIA and the host presentation space were updated.

Query_Session_Status

Prerequisite Calls

The **Query_Session_Status** function returns various status information from the host session, or from the currently connected session if *session_id* is blank.

None.

Supplied Syntax

The syntax for the **Query_Session_Status** function is as follows:

HLLAPI('Query_session_**status',** *session_id* **)**

Supplied Parameters

Function name: **Query_Session_Status**.

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

Returned Parameters

Return Code	Explanation	
' '	Null. Not connected.	
data	**Position**	**Definition**
	1	Short session ID.
	2–9	Long name of session.
	10	Session type, where: D=3270 host E=3270 printer F=5250 host G=5250 printer H=ASCII
	11	Session characteristics, expressed as a binary number containing the session characteristics byte explained below: 0 EAB 1 PSS 2–7 Reserved. If bit 0 (EAB) = 0, the session has base attributes. If bit 0 (EAB) = 1, the session has extended attributes. If bit 1 (PSS) = 0, the session does not support programmed symbols. If bit 1 (PSS) = 1, the session supports programmed symbols.
	12–13	Number of rows in the host presentation space. This is a binary number and is not in display format. If the session type is E or G, the value is 0.
	14–15	Number of columns in the host presentation space. This is a binary number and is not in display format. If the session type is E or G, the value is 0.
	16-17	Host code page number, expressed as a binary number.
	18	Reserved.

Note: After you parse the last three fields (row, col, codepage) from the string to obtain their decimal values, use c2d(reverse(x)) to reverse the bytes.

Query_Sessions

Prerequisite Calls

The **Query_Sessions** function returns either a 12-byte description of each configured session or a null ('') if an error occurs.

None.

Supplied Syntax

The syntax for the **Query_Sessions** function is as follows:

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

Supplied Parameter

Function name: **Query_Sessions**.

Returned Parameters

Return Code	Explanation	
' '	Null. Error has occurred.	
data	**Position**	**Definition**
	1	Short session ID.
	2–9	Long name of session.
	10	Connection type H=host
	11–12	Presentation space size. This is a binary number and is not in display format. If the session type is a print session, the value is 0. (See “Query Sessions (10)” on page 105.)

Note: After you parse the last field (pssize) from the string to obtain its decimal values, use c2d(reverse(x)).

Query_System

Prerequisite Calls

The **Query_System** function returns either a 35-byte system configuration string or a null ('') if an error occurs.

None.

Supplied Syntax

The syntax for the **Query_System** function is as follows:

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

Supplied Parameter

Function name: **Query_System**.

Returned Parameters

Return Code	Explanation	
' '	Null. Error has occurred.	
data	**Position**	**Definition**
	1	EHLLAPI version number.
	2–3	EHLLAPI level number.
	4–9	Reserved
	10–12	Reserved.
	13	Hardware base U= unable to determine.
	14	Program type, where P =IBM Personal Communications.
	15–16	Reserved.
	17–18	PCOMM version/level as a 2-byte ASCII value.
	19	Reserved.
	20–23	Reserved
	24–27	Reserved
	28–29	Reserved
	30–31	NLS type expressed as a 2–byte binary number.
	32	1-byte printable ASCII code representing the type of monitor used as follows: V = VGA H = XGA U = unknown
	33–35	Reserved.

Query_Window_Coord

Prerequisite Calls

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

Connect_PM

Supplied Syntax

The syntax for the **Query_Window_Coord** function is as follows:

HLLAPI('Query_window_**coord',** *session_id* **)**

Supplied Parameters

Function name: **Query_Window_Coord**.

*session_id***:** The single-character short name of the session window.

Returned Parameters

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

Receive_File

Prerequisite Calls

The **Receive_File** function is used to transfer a file from the host session to the workstation session.

ATTENTION

Do not terminate the Receive program while file transfer is in progress; otherwise you will receive an error message.

None.

Supplied Syntax

The syntax for the **Receive_File** function is as follows:

HLLAPI('Receive_file', *string* **)**

Supplied Parameters

Function name: **Receive_File**.

*string***:** The same parameters that are specified with RECEIVE command. For additional information on parameters, refer to “Receive File (91)” on page 114.

Returned Parameters

Return Code	Explanation
2	A parameter error occurred, or you specified a data string length that is too short or too long (0 bytes or more than 128 bytes) for the EHLLAPI buffer. The file transfer was unsuccessful.
3	The file transfer was complete.
4	The file transfer was complete and has segmented records.
9	A system error occurred.
27	Either the file transfer ended by a cancel of a file transfer or, if a timeout was specified by the **Set_Session_Parms** function, the timeout expired.
101	File transfer was successful (transfer to/from CICS).
300+x	The Win32 error codes reported by EHLLAPI are greater than 300. To determine the Win32 error code, subtract 300 from the return code and refer to the

Other return codes can also be received, which relate to message numbers generated by the host transfer program. For transfers to a CICS host transfer program, subtract 100 from the return code to give you the numeric portion of the message. For example, a return code of 101 would mean that the message number INW0001 was issued by the host. For other host transfer programs, just use the return code as the numerical part of the message. For example, a return of 34 would mean that message TRANS34 was issued by the host transfer program. The documentation for your host transfer program should give more information about the meanings of the specific messages.

Release

Prerequisite Calls

The **Release** function unblocks the connected display session keyboard.

Connect

Supplied Syntax

The syntax for the **Release** function is as follows:

```
HLLAPI( 'Release' )
```

Supplied Parameter

Function name: **Release**.

Returned Parameters

Return Code	Explanation
0	The **Release** function was successful.
1	Your program is not currently connected to the host session.
9	A system error occurred.

Additional Information

If you disconnect while the keyboard is locked (through the **Reserve** function), the keyboard is released automatically.

Reserve

Prerequisite Calls

The **Reserve** function blocks the currently connected session from user input until either a **Release** or a **Disconnect** function is executed.

Connect

Supplied Syntax

The syntax for the **Reserve** function is as follows:

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

Supplied Parameter

Function name: **Reserve**.

Returned Parameters

Return Code	Explanation
0	The **Reserve** function was successful.
1	Your program is not currently connected to the host session.
5	The presentation space was inhibited.
9	A system error occurred.

Reset_System

Prerequisite Calls

The **Reset_System** function reinitializes the session parameters (set by the **Set_Session_Parms** function) to their defaults and disconnects from all connected resources.

None.

Supplied Syntax

The syntax for the **Reset_System** function is as follows:

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

Supplied Parameter

Function name: **Reset_System**.

Returned Parameters

Return Code	Explanation
0	The **Reset_System** function was successful.
9	A system error occurred.

Search_Field

Prerequisite Calls

The **Search_Field** function searches the currently connected presentation space for the occurrence of a specified string beginning at a particular target field. If the SRCHALL (default) option is specified in the **Set_Session_Parms** function, the *pos* parameter is overridden.

Connect

Supplied Syntax

The syntax for the **Search_Field** function is as follows:

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

Supplied Parameters

Function name: **Search_Field**.

string: The string to search for.

pos: The position of the field within the presentation space.

Returned Parameters

Return Code	Explanation
0	The *string* was not found or the session was not connected.
data	The position of the *string* in the connected presentation space.

DBCS Only: If the specified start position for the search function is the second byte of the double-byte character, the search starts from the next character for SRCHRFWD or from this character for SRCHBKWD. If the last character of the specified string is the first byte of the double-byte character, it is not included.

During a search, SO/SI pairs are ignored in the presentation space. To search the control character of the double-byte character, the string should be placed between SO (X'0E') and SI (X'0I'). For example, X'0E000C0F' in the data string is treated as the double-byte character FF (X'000C').

Search_PS

Prerequisite Calls

The **Search_PS** function searches the host presentation space for a particular string.

Connect

Supplied Syntax

The syntax for the **Search_PS** function is as follows:

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

Supplied Parameters

Function name: **Search_PS**.

string: The string to search for.

pos: The PS position to begin the search.

Returned Parameters

Return Code	Explanation
0	The *string* was not found or the session was not connected.
data	Position of the *string* in the connected presentation space.

DBCS Only: If the specified start position for the search function is the second byte of the double-byte character, the search starts from the next character for SRCHRFWD or from this character for SRCHBKWD. If the last character of the specified string is the first byte of the double-byte character, it is not included.

During a search, SO/SI pairs are ignored in the presentation space. To search the control character of the double-byte character, the string should be placed between SO (X'0E') and SI (X'0I'). For example, X'0E000C0F' in the data string is treated as the double-byte character FF (X'000C').

Send_File

Prerequisite Calls

The **Send_File** function transfers a file from the personal computer session where EHLLAPI is running to a host session.

ATTENTION

Do not terminate the Send program while file transfer is in progress; otherwise you will receive an error message.

None.

Supplied Syntax

The syntax for the **Send_File** function is as follows:

HLLAPI('Send_file', *string* **)**

Supplied Parameters

Function name: **Send_File**.

string The same parameters that are specified with SEND command. For additional information on parameters, refer to “Send File (90)” on page 123.

Returned Parameters

Return Code	Explanation
2	A parameter error occurred, or you specified a data string length that was too long or too short for the EHLLAPI buffer. File transfer was unsuccessful.
3	The file transfer was complete.
4	The file transfer was complete and has segmented records.
5	The workstation file name was incorrect or was not found. File transfer was canceled.
9	A system error occurred.
27	Either the file transfer ended by a cancel of a file transfer or, if timeout was specified by the **Set_Session_Parms** function, the timeout expired.
101	File transfer was successful (transfer to/from CICS).
300+x	The Win32 error codes reported by EHLLAPI are greater than 300. To determine the Win32 error code, subtract 300 from the return code and refer to the

Other return codes can also be received, which relate to message numbers generated by the host transfer program. For transfers to a CICS host transfer program, subtract 100 from the return code to give you the numeric portion of the message. For example, a return code of 101 would mean that the message number INW0001 was issued by the host. For other host transfer programs, just use the return code as the numerical part of the message. For example, a return of 34 would mean that message TRANS34 was issued by the host transfer program. The documentation for your host transfer program should give more information about the meanings of the specific messages.

Sendkey

Prerequisite Calls

The **Sendkey** function sends a keystroke or a string of keystrokes to the currently connected host presentation space. The *string* parameter defines the set of keystrokes, which are sent to the host presentation space. Up to 255 keys can be sent at a time.

Connect

Supplied Syntax

The syntax for the **Sendkey** function is as follows:

HLLAPI('Sendkey', *string* **)**

Supplied Parameters

Function name: **Sendkey**.

*string***:** The string of keystrokes. Refer to “Keyboard Mnemonics” on page 128.

Returned Parameters

Return Code	Explanation
0	The keystrokes were sent; status was normal.
1	Your program is not currently connected to the host session.
4	The host session was busy; all of the keystrokes could not be sent.
5	Input to the target session was inhibited; keystrokes were rejected, or incorrect keystroke mnemonics were sent. All of the keystrokes could not be sent.
6	Bad keystroke mnemonic.
9	A system error occurred.

Set_Cursor_Pos

Supplied Syntax

The **Set_Cursor_Pos** function positions the cursor at the specified target field within the currently connected host presentation space.

The syntax for the **Set_Cursor_Pos** function is as follows:

HLLAPI('Set_cursor_**pos',** *pos* **)**

Prerequisite Calls

Connect

Supplied Parameters

Function name: **Set_Cursor_Pos**.

*pos***:** The target field to be copied.

Returned Parameters

Return Code	Explanation
0	Cursor was successfully located at specified position.
1	Your program is not currently connected to the host session.

Set_Session_Parms

Prerequisite Calls

The **Set_Session_Parms** function sets the current session parameters.

None.

Supplied Syntax

The syntax for the **Set_Session_Parms** function is as follows:

HLLAPI('Set_session_**parms',** *string* **)**

Supplied Parameters

Function name: **Set_Session_Parms**.

*string***:** The string containing the session options to be changed.

Returned Parameters

Return Code	Explanation
0	The session parameters were set.
2	One or more parameters were not valid.
9	A system error occurred.

Additional Information

The STREOT and EOT options are not supported in the **Set_Session_Parms** function.

Set_Window_Status

Prerequisite Calls

The **Set_Window_Status** function changes the window status of a session.

Connect_PM

Supplied Syntax

The syntax for the **Set_Window_Status** function is as follows:

HLLAPI('Set_window_**status',** *session_id*, *option* [, *num1* | *option1*, *num2*] **)**

Supplied Parameters

Function name: **Set_Window_Status**.

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

*option***:** See the following table:

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

The *num1* and *num2* parameters are used only for the *Move* ('M') and *Size* ('S') options and the *option1* parameter is used for the *Zorder* ('Z') option.

Returned Parameters

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

Start_Close_Intercept

Prerequisite Calls

The **Start_Close_Intercept** function intercepts close requests for the host session.

None.

Supplied Syntax

The syntax for the **Start_Close_Intercept** function is as follows:

HLLAPI('Start_close_**intercept',** *session_id* **)**

Supplied Parameters

Function name: **Start_Close_Intercept**.

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

Returned Parameters

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.

Start_Host_Notify

Prerequisite Calls

The **Start_Host_Notify** function determines if the designated host presentation space or operator information area has been updated.

None.

Supplied Syntax

The syntax for the **Start_Host_Notify** function is as follows:

HLLAPI('Start_host_**notify',** *session_id*, *option* **)**

Supplied Parameters

Function name: **Start_Host_Notify**.

session_id: The single-character short name of the session.

option: See the following table:

Value	Explanation
'P'	Asks for notification of presentation space update only.
'O'	Asks for notification of OIA update only.
'B'	Asks for notification of both presentation space and OIA updates.

Returned Parameters

Return Code	Explanation
0	The **Start_Host_Notify** function was successful.
1	An incorrect host presentation space was specified.
2	An error was made in specifying parameters.
9	A system error occurred.

Start_Keystroke_Intercept

Prerequisite Calls

The **Start_Keystroke_Intercept** function filters any keystrokes sent to the session specified by the *session_id* parameter.

None.

Supplied Syntax

The syntax for the **Start_Keystroke_Intercept** function is as follows:

HLLAPI('Start_keystroke_**intercept',** *session_id*, *option* **)**

Supplied Parameters

Function name: **Start_Keystroke_Intercept**.

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

*option***:** See the following table:

Option	Explanation
'D'	AID keys only.
'L'	All keystrokes.

Returned Parameters

Return Code	Explanation
0	The **Start_Keystroke_Intercept** function was successful.
1	The presentation space was not valid.
2	An incorrect option was specified.
4	Resource was unavailable. The requested presentation space was in use by another API application.
9	A system error occurred.

Stop_Close_Intercept

Prerequisite Calls

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

Start_Close_Intercept

Supplied Syntax

The syntax for the **Stop_Close_Intercept** function is as follows:

HLLAPI('Stop_close_**intercept',** *session_id* **)**

Supplied Parameters

Function name: **Stop_Close_Intercept**.

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

Returned Parameters

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

Stop_Host_Notify

Prerequisite Calls

The **Stop_Host_Notify** function prevents the **Start_Host_Notification** function from determining if the host session identifier has been updated.

Start_Host_Notify

Supplied Syntax

The syntax for the **Stop_Host_Notify** function is as follows:

HLLAPI('Stop_host_**notify',** *session_id* **)**

Supplied Parameters

Function name: **Stop_Host_Notify**.

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

Returned Parameters

Return Code	Explanation
0	The **Stop_Host_Notify** function was successful.
1	An incorrect host presentation space was specified.
8	No prior **Start_Host_Notify** function was issued.
9	A system error occurred.

Stop_Keystroke_Intercept

Prerequisite Calls

The **Stop_Keystroke_Intercept** function ends your application program's ability to intercept keystrokes for the *session_id*. The **Start_Keystroke_Intercept** function cancels the preceding **Start_Keystroke_Intercept** function.

Start_Keystroke_Intercept

Supplied Syntax

The syntax for the **Stop_Keystroke_Intercept** function is as follows:

HLLAPI('Stop_keystroke_**intercept',** *session_id* **)**

Supplied Parameters

Function name: **Stop_Keystroke_Intercept**.

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

Returned Parameters

Return Code	Explanation
0	The **Stop_Keystroke_Intercept** function was successful.
1	The host presentation space was not valid.
8	No prior **Start_Keystroke_Intercept** function was called for this presentation space.
9	A system error occurred.

Wait

Prerequisite Calls

The **Wait** function checks the status of the currently connected session. If the controller or host system is busy, this function causes EHLLAPI to wait for a specified time to see if the condition clears. The specified time is determined by the TWAIT, NWAIT, or LWAIT option in the **Set_Session_Parms** function.

Connect

Supplied Syntax

The syntax for the **Wait** function is as follows:

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

Supplied Parameter

Function name: **Wait**.

Returned Parameters

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

Programming Notes

In the REXX environment, the PC/3270 or PC400 for OS/2 EHLLAPI does not automatically disconnect and reset a connected session when the REXX application program ends. If the REXX application program ends (either normally or abnormally) without disconnecting the session, the session remains connected. In addition, any options set by the **Set_Session_Parms** function remain in effect. Therefore, it is the responsibility of the REXX programmer to ensure that the REXX program properly disconnects and resets under all program termination conditions, including error conditions.

Several EHLLAPI features are not applicable to the REXX EHLLAPI environment. Structured fields of the EHLLAPI features are not supported by REXX EHLLAPI.

Appendix H. Sample Programs

Sample programs demonstrating the use of REXX EHLLAPI features are included on the CD-ROM and diskettes.

The first sample program (QTIME.CMD) sets the system clock based on the VM system time (run this program only if connecting to a VM host session).

The second sample program (CMMACRO.CMD) records key strokes on the host system and plays them back. This function simplifies repetitive tasks.

Note: Each sample program disconnects and releases used resources before exiting.

© Copyright IBM Corp. 1989, 1997

Appendix I. Notices

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

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
500 Columbus Avenue
Thornwood, NY 10594
USA

For license inquiries regarding double-byte character set (DBCS) information, contact:

IBM World Trade Asia Corporation Commercial Relations
IBM Kamiya-cho Building, 4-3-9 Toranomon, Minato-ku
Tokyo, Japan

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

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

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

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

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

© Copyright IBM Corp. 1989, 1997

Appendix J. Trademarks

The following terms are trademarks of the IBM Corporation in the United States or other countries or both:

ACF/VTAM
Advanced Peer-to-Peer Networking
AFP
AIX
AIXwindows
APPN
AS/400
CICS
Common User Access
CUA
Global Network
IBM
IMS
MVS/XA
MVS/ESA
NetView
OS/2
OS/400
Personal System/2
Presentation ManagerPS/2
RT PC
System/370
System/390
S/390
TopView
VM/ESA
VTAM
XGA

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.

C-bus is a trademark of Corollary, Inc.

Microsoft, Windows, and the Windows 95 logo are trademarks or registered trademarks of Microsoft Corporation.

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. 1989, 1997

Glossary

This glossary includes terms and definitions from:

The *American National Standard Dictionary for Information Systems*, ANSI X3.172-1990, copyright 1990 by the American National Standards Institute (ANSI). Copies may be purchased from the American National Standards Institute, 11 West 42nd Street, New York, New York 10036. Definitions are identified by the symbol (A) after the definition.

The ANSI/EIA Standard—440-A, *Fiber Optic Terminology* Copies may be purchased from the Electronic Industries Association, 2001 Pennsylvania Avenue, N.W., Washington, DC 20006. Definitions are identified by the symbol (E) after the definition.

The *Information Technology Vocabulary* developed by Subcommittee 1, Joint Technical Committee 1, of the International Organization for Standardization and the International Electrotechnical Commission (ISO/IEC JTC1/SC1). Definitions of published parts of this vocabulary are identified by the symbol (I) after the definition; definitions taken from draft international standards, committee drafts, and working papers being developed by ISO/IEC JTC1/SC1 are identified by the symbol (T) after the definition, indicating that final agreement has not yet been reached among the participating National Bodies of SC1.

The *IBM Dictionary of Computing*, New York: McGraw-Hill, 1994.

Internet Request for Comments: 1208, *Glossary of Networking Terms*

Internet Request for Comments: 1392, *Internet Users' Glossary*

The *Object-Oriented Interface Design: IBM Common User Access Guidelines*, Carmel, Indiana: Que, 1992.

The following cross-references are used in this glossary:

Contrast with: This refers to a term that has an opposed or substantively different meaning.
Synonym for: This indicates that the term has the same meaning as a preferred term, which is defined in its proper place in the glossary.
Synonymous with: This is a backward reference from a defined term to all other terms that have the same meaning.
See: This refers the reader to multiple-word terms that have the same last word.
See also: This refers the reader to terms that have a related, but not synonymous, meaning.
Deprecated term for: This indicates that the term should not be used. It refers to a preferred term, which is defined in its proper place in the glossary.

A

abend. Abnormal end of task; the termination of a task before its completion because of an error condition that cannot be resolved by recovery facilities while the task is executing.

accept. (1) In a VTAM application program, to establish a session with a logical unit (LU) in response to a CINIT request from a system services control point (SSCP). The session-initiation request may begin when a terminal user logs on, a VTAM application program issues a macroinstruction, or a VTAM operator issues a command. (2) An SMP process that moves distributed code and MVS-type programs to the distribution libraries.

ACCESS. In the Simple Network Management Protocol (SNMP), the clause in a Management Information Base (MIB) module that defines the minimum level of support that a managed node provides for an object.

action. (1) An operation on a managed object, the semantics of which are defined as part of the managed object class definition. (2) In the AIX operating system, a defined task that an application performs. An action modifies the properties of an object or manipulates the object in some way.

adapter. (1) A hardware component that must be installed in the personal computer to connect to the SDLC, LAN, asynchronous, DFT, or other communication attachment (possibly connecting through a modem). (2) A part that electrically or physically connects a device to a computer or to another device.

Advanced Peer-to-Peer Networking (APPN). An extension to SNA featuring (a) greater distributed network control that avoids critical hierarchical dependencies, thereby isolating the effects of single points of failure; (b) dynamic exchange of network topology information to foster ease of connection, reconfiguration, and adaptive route selection; (c) dynamic definition of network resources; and (d) automated resource registration and directory lookup. APPN extends the LU 6.2 peer orientation for end-user services to network control and supports multiple LU types, including LU 2, LU 3, and LU 6.2.

Advanced Peer-to-Peer Networking (APPN) end node. A node that provides a broad range of end-user services and supports sessions between its local control point (CP) and the CP in an adjacent network node. It

© Copyright IBM Corp. 1989, 1997

uses these sessions to dynamically register its resources with the adjacent CP (its network node server), to send and receive directory search requests, and to obtain management services. An APPN end node can also attach to other end nodes.

Advanced Peer-to-Peer Networking (APPN) node. An APPN network node or an APPN end node.

AID. Attention identifier.

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

allocate. (1) An LU 6.2 application programming interface (API) verb used to assign a session to a conversation for the conversation's use. (2) Contrast with *deallocate*.

all points addressable (APA). In computer graphics, pertaining to the ability to address and display or not display each picture element (pel) on a display surface.

AP. Alternate printer.

APA. All points addressable.

API. Application programming interface.

APL. A programming language requiring the use of a special keyboard to represent various operator symbols.

application. A collection of software components used to perform specific types of user-oriented work on a computer.

application program. (1) A program written for or by a user that applies to the user's work, such as a program that does inventory control or payroll. (2) A program used to connect and communicate with stations in a network, enabling users to perform application-oriented activities.

Apply. A push button that carries out the selected choices in a window without closing the window.

APPN. Advanced Peer-to-Peer Networking.

argument. A parameter passed between a calling program and a called program.

ASCII (American National Standard Code for Information Interchange). The standard code, using a coded character set consisting of 7-bit coded characters (8 bits including parity check), that is used for information interchange among data processing systems, data communication systems, and associated equipment. The ASCII set consists of control characters and graphic characters. (A)

asynchronous (ASYNC). (1) Pertaining to two or more processes that do not depend upon the occurrence of specific events such as common timing signals. (T) (2) Without regular time relationship; unexpected or unpredictable with respect to the execution of program instructions.

asynchronous request. In VTAM, a request for an asynchronous operation. Contrast with *synchronous request*.

atom. A character string that has been reduced to a unique integer value.

attention identifier (AID). A character in a data stream indicating that the user has pressed a key, such as the Enter key, that requests an action by the system.

attribute. Variable data that is logically a part of an object and that represents a property of the object. For example, a serial number is an attribute of an equipment object.

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

B

Basic Input/Output System (BIOS). Code that controls basic hardware operations, such as interactions with diskette drives, hard disk drives, and the keyboard.

batch. (1) An accumulation of data to be processed. (2) A group of records or data processing jobs brought together for processing or transmission. (3) Pertaining to activity involving little or no user action. Contrast with *interactive*.

behavior. (1) Ideally, a collection of assertions that describe the allowed states that a managed object can assume. An assertion can be a precondition, a postcondition, or an invariant. In practice, the behavior is often an informal description of the semantics of attributes, operations, and notifications. (2) The way in which managed objects, name bindings, attributes, notifications, and operations interact with the actual resources that they model and with each other.

binary. Pertaining to the base two system of numbers. The binary digits are 0 and 1. Executable files are generally in binary format rather than the character string format that text files are composed of.

BIND. In SNA, a request to activate a session between two logical units (LUs). See also *session activation request*. Contrast with *UNBIND*.

BIOS. (1) Basic Input/Output System. (2) See also *NetBIOS*.

bit. Either of the digits 0 or 1 when used in the binary numeration system. (T)

bracket protocol. In SNA, a data flow control protocol in which exchanges between two session partners are achieved through the use of brackets, with one partner designated at session activation as the first speaker and the other as the bidder. The bracket protocol involves bracket initiation and termination rules.

buffer. (1) A routine or storage used to compensate for a difference in rate of flow of data, or time of occurrence of events, when transferring data from one device to another. (A) (2) A portion of storage used to hold input or output data temporarily.

byte. (1) A string that consists of a number of bits, treated as a unit, and representing a character. (T) (2) A binary character operated upon as a unit and usually shorter than a computer word. (A) (3) A group of 8 adjacent binary digits that represent one EBCDIC character.

C

call. (1) The action of bringing a computer program, a routine, or a subroutine into effect, usually by specifying the entry conditions and jumping to an entry point. (I) (A) (2) In data communication, the actions necessary to make a connection between two stations on a switched line. (3) In communications, a conversation between two users. (4) To transfer control to a procedure, program, routine, or subroutine. (5) To attempt to contact a user, regardless of whether the attempt is successful.

calling. (1) The process of transmitting selection signals in order to establish a connection between data stations. (I) (A) (2) In X.25 communications, pertaining to the location or user that makes a call.

Cancel. A push button that removes a window without applying any changes made in that window.

Caps Lock. The personal computer keyboard mode entered after the Caps Lock key is pressed. This mode is indicated by a capital A in the operator information area (OIA). When the Shift key is pressed while in this mode, Caps Lock is temporarily discontinued and the OIA indicator changes. Pressing the Shift key again returns the Caps Lock mode.

case-sensitive. Pertaining to the ability to distinguish between uppercase and lowercase letters.

character cell. The maximum physical boundary of a character on a display screen.

character set. A finite group of characters defined for a keyboard or output device.

CICS. Customer Information Control System.

C language. A language used to develop software applications in compact, efficient code that can be run on different types of computers with minimal change.

class. (1) In object-oriented design or programming, a group of objects that share a common definition and that therefore share common properties, operations, and behavior. Members of the group are called instances of the class. (2) In the AIX operating system, pertaining to the I/O characteristics of a device. System devices are classified as block or character devices.

client. (1) A functional unit that receives shared services from a server. (T) (2) A user. (3) In an AIX distributed file system environment, a system that is dependent on a server to provide it with programs or access to programs. (4) Synonymous with *requester*.

clipboard. An area of storage provided by the system to hold data temporarily.

Close. A choice that removes a window and all of the windows associated with it from the workplace. For example, if a user is performing a task in a window and a message appears, or the user asks for help, both the message and the help windows disappear when the user closes the original window.

CMS. Conversational monitor system.

coaxial cable. A cable consisting of one conductor, usually a small copper tube or wire, within and insulated from another conductor of larger diameter, usually copper tubing or copper braid.

code page. (1) A table that defines a coded character set by assignment of a character meaning to each code point in the table for a language or a country. (2) A mapping between characters and their internal binary representation. (3) An assignment of graphic characters and control function meanings to all code points; for example, assignment of characters and meanings to 256 code points for an 8-bit code, assignment of characters and meanings to 128 code points for a 7-bit code. (4) In the Print Management Facility, a font library member that associates code points and character identifiers. A code page also identifies invalid code points. (5) A particular assignment of hexadecimal identifiers to graphic characters. (6) In AFP support, a font file that associates code points and graphic character identifiers.

command. (1) A request from a terminal for the performance of an operation or the execution of a particular program. (2) In SNA, any field set in the transmission header (TH), request header (RH), and sometimes portions of a request unit (RU), that initiates an action or that begins a protocol; for example: (a)

Bind Session (session-control request unit), a command that activates an LU-LU session, (b) the change-direction indicator in the RH of the last RU of a chain, (c) the virtual route reset window indicator in an FID4 transmission header.

command list. In the NetView program, a list of commands and statements designed to perform a specific function for the user. Command lists can be written in REXX or in the NetView command list language.

command name. The first term in a command, usually followed by operands.

command prompt. A displayed character or string of characters that indicates that a user may enter a command to be processed.

Communications Manager/2. See *Communications Server* and *Personal Communications product family*. The function of the Communications Manager/2 product has been incorporated into the Communications Server product and the Personal Communications product family.

compile. (1) To translate all or part of a program expressed in a high-level language into a computer program expressed in an intermediate language, an assembly language, or a machine language. (T) (2) To prepare a machine language program from a computer program written in another programming language by making use of the overall logic structure of the program, or generating more than one computer instruction for each symbolic statement, or both, as well as performing the function of an assembler. (A) (3) To translate a source program into an executable program (an object program). (4) To translate a program written in a high-level programming language into a machine language program.

compiler. (1) A program that translates a source program into an executable program (an object program). (2) A program that decodes instructions written as pseudo codes and produces a machine language program to be executed at a later time.

component. Hardware or software that is part of a functional unit.

configuration. (1) The manner in which the hardware and software of an information processing system are organized and interconnected. (T) (2) The devices and programs that make up a system, subsystem, or network. (3) In Personal Communications, the arrangement of personal computers connected to one or more host systems by one or more attachment types. Examples are: SDLC, LAN, ASYNCH, X.25, or DFT.

configure. To describe to a system the devices, optional features, and programs installed on the system.

conjunction. The Boolean operation whose result has the Boolean value 1 if and only if each operand has the Boolean value 1. (I) (A)

connected. In VTAM, the state of a physical unit (PU) or a logical unit (LU) that has an active physical path to the host processor containing the system services control point (SSCP) that controls the respective PU or LU.

connection. (1) In data communication, an association established between functional units for conveying information. (I) (A) (2) In SNA, the network path that links together two logical units (LUs) in different nodes to enable them to establish communications. (3) In TCP/IP, the path between two protocol applications that provides reliable data stream delivery service. In the Internet, a connection extends from a TCP application on one system to a TCP application on another system.

connectivity. (1) The capability of a system or device to be attached to other systems or devices without modification. (T) (2) The capability to attach a variety of functional units without modifying them.

control block. (1) A storage area used by a computer program to hold control information. (I) (2) In the IBM Token-Ring Network, a specifically formatted block of information provided from the application program to the Adapter Support Interface to request an operation.

control character. A character whose occurrence in a particular context specifies a control function. (T)

controller. A device that coordinates and controls the operation of one or more input/output devices, such as workstations, and synchronizes the operation of such devices with the operation of the system as a whole.

control point (CP). (1) A component of an APPN or LEN node that manages the resources of that node. In an APPN node, the CP is capable of engaging in CP-CP sessions with other APPN nodes. In an APPN network node, the CP also provides services to adjacent end nodes in the APPN network. (2) A component of a node that manages resources of that node and optionally provides services to other nodes in the network. Examples are a system services control point (SSCP) in a type 5 subarea node, a network node control point (NNCP) in an APPN network node, and an end node control point (ENCP) in an APPN or LEN end node. An SSCP and an NNCP can provide services to other nodes.

Control Program (CP). In VM/ESA, a component that manages the resources of a single computer so multiple computing systems appear to exist. Each of these apparent systems, or virtual machines, is the functional

equivalent of an IBM System/370, 370-XA, or ESA computer.

control unit. A device that manages the flow of data between personal computers used with Personal Communications and a host system.

control unit terminal (CUT) mode. An IBM protocol used for communications with an IBM 3174 or 3274 Control Unit or other appropriate interface unit. In this protocol, a program in the workstation emulates a 3278 or 3879 terminal for a user at a virtual terminal, and the interface unit is responsible for enforcing the protocol.

conversation. A logical connection between two transaction programs using an LU 6.2 session. Conversations are delimited by brackets to gain exclusive use of a session.

conversational monitor system (CMS). A virtual machine operating system that provides general interactive time sharing, problem solving, and program development capabilities, and operates only under control of the VM control program.

country code. In X.25 communications, the 3-digit number that precedes the national terminal number in the network user address for public networks.

CP. (1) Control point. (2) In VM, Control Program.

CS. Current state.

Customer Information Control System (CICS). An IBM licensed program that enables transactions entered at remote terminals to be processed concurrently by user-written application programs. It includes facilities for building, using, and maintaining databases.

customization. The process of configuring one or more personal computers connected to a host system by the SDLC, LAN, ASYNCH, X.25, DFT, or other attachment. Customization precedes Personal Communications installation and includes the definition of the sessions to be carried out by Personal Communications.

D

data stream. (1) All information (data and control commands) sent over a data link usually in a single read or write operation. (2) A continuous stream of data elements being transmitted, or intended for transmission, in character or binary-digit form, using a defined format.

DBCS. Double-byte character set.

definition file. A file loaded by default that contains keyboard and screen color characteristics that can be adapted for a particular session. The lines of a definition file must be coded according to a strict layout.

delimiter. (1) A character used to indicate the beginning and end of a character string. (T) (2) A flag that separates and organizes items of data. (A) (3) A character that groups or separates words or values in a line of input. (4) In the IBM Token-Ring Network, a bit pattern that defines the limits of a frame or token.

destination. (1) Any point or location, such as a node, station, or a particular terminal, to which information is to be sent. (2) An external logical unit (LU) or application program to which messages or other data are directed.

device. A mechanical, electrical, or electronic contrivance with a specific purpose.

device driver. (1) A file that contains the code needed to use an attached device. (2) A program that enables a computer to communicate with a specific peripheral device; for example, a printer, a videodisc player, or a CD drive. (3) A collection of subroutines that control the interface between I/O device adapters and the processor.

dialog. (1) The interaction between a user and a computer. (2) In an interactive system, a series of related inquiries and responses similar to a conversation between two people. (3) In the AIXwindows Toolkit, a two-way text interface between an application and its user. The interface takes the form of a collection of widgets and gadgets, including a DialogShell widget, a BulletinBoard widget (or a subclass of a BulletinBoard widget or some other container widget), plus various children, including Label, PushButton, and Text widgets.

directory. (1) A table of identifiers and references to the corresponding items of data. (I) (A) (2) A named hierarchical grouping of files in a file system. (3) A database in an APPN node that lists names of resources (in particular, logical units) and records the CP name of the node where each resource is located.

distributed processing. Processing that takes place across two or more linked systems.

DLL. Dynamic link library.

DOS. Disk Operating System.

DOS session. A session in which a personal computer operates as a stand-alone computer, running under Disk Operating System (DOS). See *host session*.

double-byte character set (DBCS). A set of characters in which each character is represented by 2 bytes. Languages such as Japanese, Chinese, and

Korean, which contain more symbols than can be represented by 256 code points, require double-byte character sets. Because each character requires 2 bytes, the typing, display, and printing of DBCS characters requires hardware and programs that support DBCS. Contrast with *single-byte character set (SBCS)*.

drop. In the IBM Cabling System, a cable that runs from a faceplate to the distribution panel in a wiring closet. When the IBM Cabling System is used with the IBM Token-Ring Network, a drop may form part of a lobe. Cables between wiring closets are not classified as drops.

dynamic. (1) In programming languages, pertaining to properties that can only be established during the execution of a program; for example, the length of a variable-length data object is dynamic. (I) (2) Pertaining to an operation that occurs at the time it is needed rather than at a predetermined or fixed time.

dynamic data exchange (DDE). A protocol that allows applications to exchange data.

dynamic linking. In the OS/2 operating system, the delayed connection of a program to a routine until load time or run time.

dynamic link library (DLL). A file containing executable code and data bound to a program at load time or run time, rather than during linking. The code and data in a dynamic link library can be shared by several applications simultaneously.

E

EBCDIC. Extended binary-coded decimal interchange code. A coded character set of 256 8-bit characters.

ECF. Enhanced Connectivity Facility.

EGA. Enhanced graphics adapter.

EHLLAPI. Emulator High-Level Language Application Programming Interface.

element. (1) A field in the network address. (2) In SNA, the particular resource within a subarea that is identified by an element address. See also *subarea*.

Emulation Program (EP). (1) An IBM control program that allows a channel-attached IBM communication controller to emulate the functions of an IBM 2701 Data Adapter Unit, an IBM 2702 Transmission Control, or an IBM 2703 Transmission Control. (2) See also *network control program*.

emulator. A program that allows a device to operate as if it were a different type of device. Personal Communications, for example, allows supported personal computers and printers to operate as if they were 3270-series workstations.

Emulator High-Level Language Application Programming Interface (EHLLAPI). In Communications Manager/2, an application programming interface that provides programming access to the area in computer memory that corresponds to the user's screen image (this area in memory is known as the “presentation space”).

Enhanced Connectivity Facility (ECF). (1) A set of programs used for interconnecting IBM personal computers and IBM System/370 or System/390 host computers operating in the MVS/XA or VM/ESA environment. These ECF programs provide a method for sharing resources between workstations and host systems. (2) An IBM communication utility that provides the ability to share resources between workstations and host systems.

enhanced graphics adapter (EGA). An adapter, such as the IBM Enhanced Graphics Adapter, that provides high-resolution graphics, allowing the use of a color display for text processing as well as graphics applications.

entry point (EP). (1) The address or label of the first instruction executed on entering a computer program, routine, or subroutine. A computer program, routine, or subroutine may have a number of different entry points, each perhaps corresponding to a different function or purpose. (I) (A) (2) In SNA, a type 2.0, type 2.1, type 4, or type 5 node that provides distributed network management support. It sends network management data about itself and the resources it controls to a focal point for centralized processing, and it receives and executes focal-point initiated commands to manage and control its resources.

EOF. End of Field

EOT. End of Text

ER. Explicit route.

event. An occurrence of significance to a task; for example, an SNMP trap, the opening of a window or a submap, or the completion of an asynchronous operation.

exception response (ER). In SNA, a protocol requested in the form-of-response-requested field of a request header that directs the receiver to return a response only if the request is unacceptable as received or cannot be processed; that is, a negative response, but not a positive response, can be returned. Contrast with *definite response* and *no response*.

exit list. In the OS/2 operating system, a list of subroutines that receive control from the base operating system when a particular process ends, either normally or abnormally.

EXT. External trace file.

extended binary-coded decimal interchange code (EBCDIC). The standard code, using a character set consisting of 8-bit coded characters, used by Personal Communications for information interchange between personal computers and a host system.

extended error code. An 8-byte data string returned by Query System generated by an internal system error that is used by service personnel for diagnosis.

F

fault. An accidental condition that causes a functional unit to fail to perform its required function. (I) (A)

feature. A part of an IBM product that may be ordered separately by the customer.

field. (1) An area in a record or panel used to contain data. (2) In the IBM 3270 data stream, a group of consecutive positions on a presentation space having similar characteristics that are defined by a field attribute byte at the beginning of the field. (3) An identifiable area in a window. Examples of fields are: an entry field, into which a user can type or place text, and a field of radio button choices, from which a user can select one choice.

file transfer. The transfer of one or more files from one system to another over a data link.

file type. In VM/CMS, the second field of the file identifier. The file type can be arbitrary, or of significance to the system (for example, SCRIPT, LIST3270, and NOTEBOOK file types).

filter. A device or program that separates data, signals, or material in accordance with specified criteria. (A)

fixed session-level pacing. A form of session-level pacing in which the data transfer rate is controlled using fixed pacing-window sizes, which are initialized at session-activation time.

flag. (1) To mark an information item for selection for further processing. (T) (2) A character that signals the occurrence of some condition, such as the end of a word. (A) (3) A character or bit sequence that marks an occurrence or boundary, such as the end of a word or the beginning or end of a data transmission block.

flow. In NetDA/2, the amount of traffic that can pass through a node, connection, or route in both directions during a given period of time.

frame. (1) In Open Systems Interconnection architecture, a data structure pertaining to a particular area of knowledge and consisting of slots that can accept the values of specific attributes and from which inferences can be drawn by appropriate procedural attachments. (T) (2) The unit of transmission in some local area networks, including the IBM Token-Ring Network. It includes delimiters, control characters, information, and checking characters. (3) In SDLC, the vehicle for every command, every response, and all information that is transmitted using SDLC procedures. (4) A data structure (data frame) composed of fields meeting the field specifications of a type of communication protocol. Frames are used to control data transfer across a data link. (5) In SDLC, a sequence of bits delimited by an opening and closing flag. In X.25 packet switching data networks, frames are composed of 8-bit byte sequences delimited by beginning and ending flags; the frames in X.25 control various functions, data transfer, and transmission checking.

function call. An expression that moves the path of execution from the current function to a specified function and evaluates to the return value provided by the called function. A function call contains the name of the function to which control moves and a parenthesized list of values.

H

handle. In the Advanced DOS and OS/2 operating systems, a binary value created by the system that identifies a drive, directory, and file so that the file can be found and opened.

HD. Half-duplex.

header. (1) System-defined control information that precedes user data. (2) The portion of a message that contains control information for the message such as one or more destination fields, name of the originating station, input sequence number, character string indicating the type of message, and priority level for the message.

header file. Synonym for *include file*.

hexadecimal. (1) Pertaining to a selection, choice, or condition that has 16 possible different values or states. (I) (2) Pertaining to a fixed-radix numeration system, with radix of 16. (I) (3) Pertaining to a system of numbers to the base 16; hexadecimal digits range from 0 through 9 and A through F, where A represents 10 and F represents 15.

high-level language (HLL). A programming language that does not reflect the structure of any particular computer or operating system.

high-level language application programming interface (HLLAPI). (1) A software product that supports interaction between a host program and an application program running on a personal computer in terminal emulation (usually 3270 terminal emulation). (2) A programming interface that usually operates in conjunction with an emulator, such as 3270 emulation, and allows interaction using 3270 data stream between a host and a remote application program.

hiragana. One of the two common Japanese phonetic alphabets (the other is katakana). In hiragana, each character is represented by 1 byte. See also *kanji*.

HLLAPI. High-level language application programming interface.

host. (1) In the Internet suite of protocols, an end system. The end system can be any workstation; it does not have to be a mainframe. (2) See *host processor*.

host code page. The character code page used by the host system to which a gateway connects.

host print. A printer session on a local personal computer directed by the host system.

host session. A logical connection that enables a personal computer to communicate with a host system. A session can be identified by LU address, LT number, or session ID. See *DOS session*. See also *logical terminal*.

host system. In Personal Communications, the computer linked to one or more personal computers by the SDLC, LAN, ASYNCH, X.25, or DFT attachment.

I

ID. (1) Identifier. (2) Identification.

inbound. In communications, data that is received from the network.

include file. A text file that contains declarations used by a group of functions, programs, or users. Synonymous with *header file*.

information (I) format. A format used for information transfer.

information (I) frame. A frame in I format used for numbered information transfer.

inhibited. In VTAM, pertaining to a logical unit (LU) that has indicated to its system services control point (SSCP) that it is temporarily not ready to establish LU-LU sessions. An initiate request for a session with an inhibited LU will be rejected by the SSCP. The LU can separately indicate whether this applies to its ability to act as a primary logical unit (PLU) or a secondary logical unit (SLU).

INITIATE. A network services request sent from a logical unit (LU) to a system services control point (SSCP) requesting that an LU-LU session be established.

input/output (I/O). (1) Pertaining to input, output, or both. (A) (2) Pertaining to a device, process, or channel involved in data input, data output, or both.

INT. Internal trace table.

interactive. Pertaining to the exchange of information between a user and a computer.

interface. (1) A shared boundary between two functional units, defined by functional characteristics, signal characteristics, or other characteristics, as appropriate. The concept includes the specification of the connection of two devices having different functions. (T) (2) Hardware, software, or both, that links systems, programs, or devices.

interrupt. A suspension of a process, such as execution of a computer program caused by an external event, and performed in such a way that the process can be resumed. (A)

I/O. Input/output.

J

JISCII. Japanese Industry Standard Code for Information Interchange. A code used by IBM Personal Computers in Japan. It contains Japanese Industry Standard (JIS) C 6226 “Kanji code for exchanging information” (JIS Kanji set) and user unique characters.

jump. To switch to the logical terminal session specified.

K

katakana. One of the two common Japanese phonetic alphabets (the other is hiragana). In katakana, each character is represented by 1 byte. Katakana is primarily used to write foreign words phonetically. See also *kanji*.

keylock. In Communications Manager/2, a function that controls access to configuration data and some

advanced functions. See also *lock*, *master key*, and *service key*.

keyword. (1) In programming languages, a lexical unit that, in certain contexts, characterizes some language construct; for example, in some contexts, IF characterizes an if-statement. A keyword normally has the form of an identifier. (I) (2) One of the predefined words of an artificial language. (A) (3) A significant and informative word in a title or document that describes the content of that document.

L

layer. (1) In network architecture, a group of services that is complete from a conceptual point of view, that is one out of a set of hierarchically arranged groups, and that extends across all systems that conform to the network architecture. (T) (2) In the Open Systems Interconnection reference model, one of seven conceptually complete, hierarchically arranged groups of services, functions, and protocols, that extend across all open systems. (T) (3) In SNA, a grouping of related functions that are logically separate from the functions in other groups. Implementation of the functions in one layer can be changed without affecting functions in other layers.

LEN. Low-entry networking.

line. The portion of a data circuit external to data circuit-terminating equipment (DCE), that connects the DCE to a data switching exchange (DSE), that connects a DCE to one or more other DCEs, or that connects a DSE to another DSE. (I)

link. (1) The combination of the link connection (the transmission medium) and two link stations, one at each end of the link connection. A link connection can be shared among multiple links in a multipoint or token-ring configuration. (2) To interconnect items of data or portions of one or more computer programs: for example, the linking of object programs by a linkage editor, linking of data items by pointers. (T)

link connection. The physical equipment providing two-way communication between one link station and one or more other link stations; for example, a telecommunication line and data circuit-terminating equipment (DCE).

link level. A part of Recommendation X.25 that defines the link protocol used to get data into and out of the network across the full-duplex link connecting the subscriber's machine to the network node. LAP and LAPB are the link access protocols recommended by the CCITT.

load-time dynamic linking. A mechanism that enables program modules and their external references to dynamic-link libraries to be loaded into memory when the program is loaded.

local. (1) Pertaining to a device accessed directly without use of a telecommunication line. (2) Contrast with *remote*. (3) Synonym for *channel-attached*.

local copy. A printer session initiated by a personal computer user or host that reproduces a presentation space with all supported graphics.

lock. (1) The means by which integrity of data is ensured by preventing more than one user from accessing or changing the same data or object at the same time. (2) In Communications Manager/2, a password-protection system that can be used to prevent access to some advanced functions. See also *keylock*.

logical terminal. (1) A destination with a name that is related to one or more physical terminals. (2) The definition of a specific 3270 or 5250 emulation session.

logical unit (LU). A type of network accessible unit that enables users to gain access to network resources and communicate with each other.

low-entry networking (LEN). A capability of nodes to attach directly to one another using basic peer-to-peer protocols to support multiple and parallel sessions between logical units.

low-entry networking (LEN) end node. A LEN node receiving network services from an adjacent APPN network node.

low-entry networking (LEN) node. A node that provides a range of end-user services, attaches directly to other nodes using peer protocols, and derives network services implicitly from an adjacent APPN network node, that is, without the direct use of CP-CP sessions.

LU. Logical unit.

LU-LU session. A logical connection between two logical units (LUs) in an SNA network that typically provides communication between two users.

M

maintenance analysis procedure (MAP). A maintenance document that gives an IBM service representative a step-by-step procedure for tracing a symptom to the cause of a failure.

manager. (1) In systems management, a user that, for a particular interaction, has assumed a manager role. (2) An entity that monitors or controls one or more managed objects by (a) receiving notifications regarding the objects and (b) requesting management operations

to modify or query the objects. (3) A system that assumes a manager role.

map. In NetView for AIX, a database represented by a set of related submaps that provide a graphical and hierarchical presentation of a network and its systems.

MAP. Maintenance analysis procedure.

mapping. The process of converting data that is transmitted in one format by the sender into the data format that can be accepted by the receiver.

matching. In MPTN architecture, pertaining to the relationship between peer transport users or peer transport providers that use the same user protocols or the same transport protocols.

medium. (1) A physical carrier of electrical energy. (2) A physical material in or on which data may be represented.

module. A program unit that is discrete and identifiable with respect to compiling, combining with other units, and loading; for example, the input to or output from an assembler, compiler, linkage editor, or executive routine. (A)

module definition file. In the OS/2 operating system, a file, used at link-edit time, that describes the attributes for the executable file being built (for example, load-on-call or preload attributes for segments).

monitor. (1) A device that observes and records selected activities within a data processing system for analysis. Possible uses are to indicate significant departure from the norm, or to determine levels of utilization of particular functional units. (T)
(2) Software or hardware that observes, supervises, controls, or verifies operations of a system. (A)
(3) The function required to initiate the transmission of a token on the ring and to provide soft-error recovery in case of lost tokens, circulating frames, or other difficulties. The capability is present in all ring stations.
(4) In the NetView Graphic Monitor Facility, to open a view that can receive status changes from the NetView program. Problem determination and correction can be performed directly from the view. Contrast with *browse*.

MVS. Multiple Virtual Storage. Implies MVS/390, MVS/XA, and MVS/ESA.

N

NA. Not applicable. When this appears in a calling parameter position, it means that PC400 EHLLAPI does not require this parameter to perform the specific function.

national language support (NLS). The modification or conversion of a United States English product to conform to the requirements of another language or country. This can include the enabling or retrofitting of a product and the translation of nomenclature, machine-readable information (MRI), or documentation of a product. NLS is also known as "internationalization."

native. In MPTN architecture, pertaining to the relationship between a transport user and a transport provider that are both based on the same transport protocol.

NC. Network control.

network. (1) An arrangement of nodes and connecting branches. (T) (2) A configuration of data processing devices and software connected for information interchange. (3) A group of nodes and the links interconnecting them.

network control (NC). In SNA, a request/response unit (RU) category used for requests and responses exchanged between physical units (PUs) for such purposes as activating and deactivating explicit and virtual routes and sending load modules to adjust peripheral nodes.

NLS. National language support.

NN. Network node.

no response. In SNA, a protocol requested in the form-of-response-requested field of the request header that directs the receiver of the request not to return any response, regardless of whether or not the request is received and processed successfully.

normal stop. In Communications Manager/2, stopping communications after current functions (for example, file transfer) have completed.

notification. An unscheduled, spontaneously generated report of an event that has occurred.

NOTIFY. A network services request that is sent by a system services control point (SSCP) to a logical unit (LU) to inform the LU of the status of a procedure requested by the LU.

O

object. (1) In object-oriented design or programming, an abstraction consisting of data and the operations associated with that data. See also *class*. (2) An item that a user can manipulate as a single unit to perform a task. An object can appear as text, an icon, or both.

OEM. Original equipment manufacturer.

Off. A choice that appears in the cascaded menu from the Refresh choice. It sets the refresh function to off.

offset. The number of measuring units from an arbitrary starting point in a record, area, or control block, to some other point.

OIA. Operator information area.

On. A choice that appears in a cascaded menu from the Refresh choice. It immediately refreshes the view in a window.

open. (1) A break in an electrical circuit. (2) To make an adapter ready for use.

Open. A choice that leads to a window in which users can select the object they want to open.

operand. (1) An entity on which an operation is performed. (I) (2) That which is operated upon. An operand is usually identified by an address part of an instruction. (A) (3) Information entered with a command name to define the data on which a command processor operates and to control the execution of the command processor.

operating system (OS). Software that controls the execution of programs and that may provide services such as resource allocation, scheduling, input/output control, and data management. Although operating systems are predominantly software, partial hardware implementations are possible. (T)

operation. In object-oriented design or programming, a service that can be requested at the boundary of an object. Operations include modifying an object or disclosing information about an object.

operator. (1) In a language statement, the lexical entity that indicates the action to be performed on operands. (2) A person or program responsible for managing activities controlled by a given piece of software such as MVS, the NetView program, or IMS. (3) A person who operates a device. (4) A person who keeps a system running.

operator information area (OIA). The area near the bottom of the display area where terminal or system status information is displayed.

operator station task (OST). The NetView task that establishes and maintains the online session with the network operator. There is one operator station task for each network operator who logs on to the NetView program. See *NetView-NetView task*.

original equipment manufacturer (OEM). A manufacturer of equipment that may be marketed by another manufacturer.

OS. Operating system.

P

parameter. (1) A variable that is given a constant value for a specified application and that may denote the application. (I) (A) (2) In Basic CUA architecture, a variable used in conjunction with a command to affect its result. (3) An item in a menu for which the user specifies a value or for which the system provides a value when the menu is interpreted. (4) Data passed to a program or procedure by a user or another program, namely as an operand in a language statement, as an item in a menu, or as a shared data structure.

parent. A process that has spawned a child process using the fork primitive.

path. (1) In a network, any route between any two nodes. A path may include more than one branch. (T) (2) The series of transport network components (path control and data link control) that are traversed by the information exchanged between two network accessible units.

path control (PC). The function that routes message units between network accessible units in the network and provides the paths between them. It converts the basic information units (BIUs) from transmission control (possibly segmenting them) into path information units (PIUs) and exchanges basic transmission units containing one or more PIUs with data link control. Path control differs by node type: some nodes (APPN nodes, for example) use locally generated session identifiers for routing, and others (subarea nodes) use network addresses for routing.

PC. (1) Path control. (2) Personal Communications.

physical connection. (1) A connection that establishes an electrical circuit. (2) A point-to-point or multipoint connection. (3) Synonymous with *connection*.

physical unit (PU). The component that manages and monitors the resources (such as attached links and adjacent link stations) associated with a node, as requested by an SSCP via an SSCP-PU session. An SSCP activates a session with the physical unit in order to indirectly manage, through the PU, resources of the node such as attached links. This term applies to type 2.0, type 4, and type 5 nodes only.

physical unit (PU) services. In SNA, the components within a physical unit (PU) that provide configuration services and maintenance services for SSCP-PU sessions.

PM. Presentation Manager.

pointer. (1) A data element that indicates the location of another data element. (T) (2) An identifier that indicates the location of an item of data. (A)

pop. To remove an item from the top of a pushdown list. Contrast with *push*.

POP. Post Office Protocol.

POR. Power-on reset function.

post. (1) To enter a unit of information on a record. (2) To note the occurrence of an event. (3) To add information to a record in order to keep the record current.

Post Office Protocol (POP). A protocol used for exchanging network mail and accessing mailboxes.

power-on reset (POR) function. A function that allows the user to reset all logical terminals. The key sequence for POR is Ctrl+Pause using the Enhanced or Space Saving keyboards. Otherwise, the sequence is Ctrl+F2.

Prepare. A presentation services header that flows as part of commit processing, indicating the partner has begun the first phase of the two-phase commit process.

presentation space. (1) A conceptual two-dimensional surface in storage on which data for a portion of the display surface is represented. (2) The space that contains the device-independent definition of a picture. (3) In the context of the Emulator High-Level Language Application Programming Interface (EHLLAPI), the area in computer memory that corresponds to the user's screen image. (4) The host display space for the window. It is the entire window of data, which can exceed the boundaries of a display screen's viewing area.

presentation space ID (PSID). In Communications Manager/2, synonym for *short name*.

private application. An application maintained for use only on one computer or user, that is not available on the network for other users.

problem determination. The process of determining the source of a problem; for example, a program component, machine failure, telecommunication facilities, user or contractor-installed programs or equipment, environmental failure such as a power loss, or user error.

procedure. In a programming language, a block, with or without formal parameters, whose execution is invoked by means of a procedure call. (I)

process. (1) To perform operations on data in a process. (I) (A) (2) In data processing, the course of events that occurs during the execution of all or part of a program. (T) (3) A course of the events defined by its purpose or by its effect, achieved under given conditions. (4) Any operation or combination of operations on data. (5) A function being performed or waiting to be performed.

processor. In a computer, a functional unit that interprets and executes instructions. A processor consists of at least an instruction control unit and an arithmetic and logic unit. (T)

product-set identification (PSID). (1) In SNA, a technique for identifying the hardware and software products that implement a network component. (2) A management services common subvector that transports the information described in definition (1).

programmed operator. The Emulator High-Level Language Application Programming Interface (EHLLAPI) application program that performs and monitors activities in a workstation presentation space. Contrast with *terminal operator*.

protected field. A field that holds data for viewing; the data cannot be changed (at the current screen) by the user.

protocol. (1) A set of semantic and syntactic rules that determine the behavior of functional units in achieving communication. (I) (2) In Open Systems Interconnection architecture, a set of semantic and syntactic rules that determine the behavior of entities in the same layer in performing communication functions. (T) (3) In SNA, the meanings of, and the sequencing rules for, requests and responses used for managing the network, transferring data, and synchronizing the states of network components.

PSID. Presentation space ID.

PU. Physical unit.

PUT. Program update tape.

PVC. Permanent virtual circuit.

Q

queue. (1) A list constructed and maintained so that the next data element to be retrieved is the one stored first. (T) (2) A line or list of items waiting to be processed; for example, work to be performed or messages to be displayed. (3) To arrange in or form a queue.

R

RAM. Random access memory. (A)

random access memory (RAM). A storage device into which data is entered and from which data is retrieved in a nonsequential manner.

Rapid Transport Protocol (RTP). A connection-oriented, full-duplex transport protocol for carrying session traffic over High-Performance Routing (HPR) routes.

record. A set of data treated as a unit. (T)

remote. (1) Pertaining to a system, program, or device that is accessed through a telecommunication line. (2) Synonym for *link-attached*. (3) Contrast with *local*.

requester. A computer that accesses shared network resources through a server. Synonym for *client*.

resource. Any facility of a computing system or operating system required by a job or task, and including main storage, input/output devices, the processing unit, data sets, and control or processing programs.

resource takeover. In VTAM, an action initiated by a network operator to transfer control of resources from one domain to another without breaking the connections or disrupting existing LU-LU sessions on the connection.

response. In data communication, a reply represented in the control field of a response frame. It advises the primary or combined station of the action taken by the secondary or other combined station to one or more commands.

response file. A file that contains a set of predefined answers to questions asked by a program and that is used in place of user dialog. methodology.

response time monitor. A function that measures and displays the host response time

Restructured Extended Executor (REXX). A general-purpose, procedural language for end-user personal programming, designed for ease by both casual general users and computer professionals. It is also useful for application macros. REXX includes the capability of issuing commands to the underlying operating system from these macros and procedures. Features include powerful character-string manipulation, automatic data typing, manipulation of objects familiar to people, such as words, numbers, and names, and built-in interactive debugging.

return code. (1) A code used to influence the execution of succeeding instructions. (A) (2) A value returned to a program to indicate the results of an operation requested by that program.

REXX. Restructured Extended Executor.

route. (1) An ordered sequence of nodes and transmission groups (TGs) that represent a path from an origin node to a destination node traversed by the traffic exchanged between them. (2) The path that network traffic uses to get from source to destination.

router. (1) A computer that determines the path of network traffic flow. The path selection is made from several paths based on information obtained from specific protocols, algorithms that attempt to identify the shortest or best path, and other criteria such as metrics or protocol-specific destination addresses. (2) An attaching device that connects two LAN segments, which use similar or different architectures, at the reference model network layer. (3) In OSI terminology, a function that determines a path by which an entity can be reached. (4) In TCP/IP, synonymous with *gateway*.

routing. (1) The process of determining the path to be used for transmission of a message over a network. (T) (2) The assignment of the path by which a message is to reach its destination. (3) In SNA, the forwarding of a message unit along a particular path through a network, as determined by parameters carried in the message unit, such as the destination network address in a transmission header.

routing table. A collection of routes used to direct datagram forwarding or to establish a connection. The information is passed among routers to identify network topology and destination feasibility.

RTP. Rapid Transport Protocol.

S

SBCS. Single-byte character set.

SDLC. Synchronous Data Link Control.

segment. (1) A section of cable between components or devices. A segment may consist of a single patch cable, several patch cables that are connected, or a combination of building cable and patch cables that are connected. (2) In Internet communications, the unit of transfer between TCP functions in different machines. Each segment contains control and data fields; the current byte-stream position and actual data bytes are identified along with a checksum to validate received data.

segmentation. A process by which path control (PC) divides basic information units (BIUs) into smaller units,

called BIU segments, to accommodate smaller buffer sizes in adjacent nodes. Both segmentation and segment assembly are optional PC features. The support for either or both is indicated in the BIND request and response.

semaphore. An indicator used to control access to a file; for example, in a multiuser application, a flag that prevents simultaneous access to a file.

sequence number. (1) In communications, a number assigned to a particular frame or packet to control the transmission flow and receipt of data. (2) A numerical value assigned by VTAM to each message exchanged between two nodes. The value (one for messages sent from the application program to the logical unit and another for messages sent from the logical unit to the application program) increases by one for each successive message transmitted unless it is reset by the application program with a set and test sequence numbers (STSN) indicator.

serial. (1) Pertaining to a process in which all events occur one after the other; for example, serial transmission of the bits of a character according to V24 CCITT protocol. (T) (2) Pertaining to the sequential or consecutive occurrence of two or more related activities in a single device or channel. (A) (3) Pertaining to the sequential processing of the individual parts of a whole, such as the bits of a character or the characters of a word, using the same facilities for successive parts. (A)

server. (1) A functional unit that provides shared services to workstations over a network; for example, a file server, a print server, a mail server. (T) (2) In a network, a data station that provides facilities to other stations; for example, a file server, a print server, a mail server. (A) (3) In the AIX operating system, an application program that usually runs in the background and is controlled by the system program controller. (4) In the Enhanced X-Windows Toolkit, a program that provides the basic windowing mechanism. It handles interprocess communication (IPC) connections from clients, demultiplexes graphics requests onto screens, and multiplexes input back to clients.

Server-Requester Programming Interface (SRPI). An IBM application programming interface (API) used by requester and server programs to communicate with the personal computer or host routers.

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

service point (SP). An entry point that supports applications that provide network management for resources not under the direct control of itself as an entry point. Each resource is either under the direct control of another entry point or not under the direct control of any entry point. A service point accessing these resources is not required to use SNA sessions (unlike a focal point). A service point is needed when entry point support is not yet available for some network management function.

session. (1) In network architecture, for the purpose of data communication between functional units, all the activities which take place during the establishment, maintenance, and release of the connection. (T) (2) A logical connection between two network accessible units (NAUs) that can be activated, tailored to provide various protocols, and deactivated, as requested. Each session is uniquely identified in a transmission header (TH) accompanying any transmissions exchanged during the session. (3) A logical connection between a server and a requester that was initiated by a successful request for a shared resource. See also *host session* and *DOS session*.

session ID. An alphabetic ID (*a* through *h*) assigned by Personal Communications to each session or screen. This applies to all types of host sessions and is used in file transfers. See also *short name*.

session parameters. In SNA, the parameters that specify or constrain the protocols (such as bracket protocol and pacing) for a session between two network accessible units.

session services (SS). One of the types of network services in the control point (CP) and in the logical unit (LU). These services provide facilities for an LU or a network operator to request that a control point (an ENCP, NNCP, or SSCP) assist with initiating or terminating sessions between logical units. Assistance with session termination is needed only by SSCP-dependent LUs.

shift-in character (SI). A code extension character used to terminate a sequence that has been introduced by the shift-out character to make effective the graphic characters of the standard character set. (I) (A)

shift-out character (SO). A code extension character that substitutes for the graphic characters of the standard character set an alternative set of graphic characters upon which an agreement has been arrived at or that has been designated using code extension procedures. (I) (A)

short name. (1) In Personal Communications, a character displayed in column 7 of the operator information area that shows the session ID. See also *session ID* and *operator information area*. (2) In Communications Manager/2, the one-letter name (A through Z) of the presentation space or emulation session. Synonymous with *presentation space ID (PSID)* and *short-session ID*.

short-session ID. In Communications Manager/2, synonym for *short name*.

SI. The shift-in character. (I) (A)

SN. Subarea node.

SNA. Systems Network Architecture.

SO. The shift-out character. (I) (A)

SP. Service point.

SRPI. Server-Requester Programming Interface.

SS. Session services.

SSCP. System services control point.

SSCP-LU session. In SNA, a session between a system services control point (SSCP) and a logical unit (LU). The session enables the LU to request the SSCP to help initiate LU-LU sessions.

statement. A language syntactic unit consisting of an operator, or other statement identifier, followed by one or more operands.

station. An input or output point of a system that uses telecommunication facilities; for example, one or more systems, computers, terminals, devices, and associated programs at a particular location that can send or receive data over a telecommunication line.

stream. (1) To send data from one device to another. (2) See *data stream*.

subarea node (SN). A node that uses network addresses for routing and maintains routing tables that reflect the configuration of the network. Subarea nodes can provide gateway function to connect multiple subarea networks, intermediate routing function, and boundary function support for peripheral nodes. Type 4 and type 5 nodes can be subarea nodes.

subarea PU. In SNA, a physical unit in a subarea node.

subdirectory. A directory contained within another directory in a file system hierarchy.

subsystem. A secondary or subordinate system, usually capable of operating independently of, or asynchronously with, a controlling system. (T)

SVC. Switched virtual circuit.

switched virtual circuit (SVC). An X.25 circuit that is dynamically established when needed. The X.25 equivalent of a switched line.

synchronous. (1) Pertaining to two or more processes that depend upon the occurrence of specific events such as common timing signals. (T) (2) Occurring with a regular or predictable time relationship.

Synchronous Data Link Control (SDLC). A discipline conforming to subsets of the Advanced Data Communication Control Procedures (ADCCP) of the American National Standards Institute (ANSI) and High-level Data Link Control (HDLC) of the International Organization for Standardization, for managing synchronous, code-transparent, serial-by-bit information transfer over a link connection. Transmission exchanges may be duplex or half-duplex over switched or nonswitched links. The configuration of the link connection may be point-to-point, multipoint, or loop. (I)

system. In data processing, a collection of people, machines, and methods organized to accomplish a set of specific functions. (I) (A)

system configuration. A process that specifies the devices and programs that form a particular data processing system.

system services control point (SSCP). A component within a subarea network for managing the configuration, coordinating network operator and problem determination requests, and providing directory services and other session services for users of the network. Multiple SSCPs, cooperating as peers with one another, can divide the network into domains of control, with each SSCP having a hierarchical control relationship to the physical units and logical units within its own domain.

system services control point (SSCP) domain. The system services control point, the physical units (PUs), the logical units (LUs), the links, the link stations, and all the resources that the SSCP has the ability to control by means of activation and deactivation requests.

Systems Network Architecture (SNA). The description of the logical structure, formats, protocols, and operational sequences for transmitting information units through, and controlling the configuration and operation of, networks. The layered structure of SNA allows the ultimate origins and destinations of information, that is, the users, to be independent of and unaffected by the specific SNA network services and facilities used for information exchange.

T

table. A repository for data that NetDA/2 uses to design a network. Each table contains information related to the network.

Task-Related User Exit (TRUE). A CICS module used for invoking resource managers that are outside of CICS. A TRUE is one of the components of CICS sockets.

terminal emulator. See *emulator*.

terminal operator. The user of an Emulator High-Level Language Application Programming Interface (EHLLAPI) application program. Contrast with *programmed operator*.

TERMINATE. In SNA, a request unit that is sent by a logical unit (LU) to its system services control point (SSCP) to cause the SSCP to start a procedure to end one or more designated LU-LU sessions.

TH. Transmission header.

thread. In the OS/2 operating system, the smallest unit of operation to be performed within a process.

timeout. (1) An event that occurs at the end of a predetermined period of time that began at the occurrence of another specified event. (I) (2) A time interval allotted for certain operations to occur; for example, response to polling or addressing before system operation is interrupted and must be restarted.

Time Sharing Option (TSO). An option of the MVS operating system that provides interactive time sharing from remote terminals.

token. (1) In a local area network, the symbol of authority passed successively from one data station to another to indicate the station temporarily in control of the transmission medium. Each data station has an opportunity to acquire and use the token to control the medium. A token is a particular message or bit pattern that signifies permission to transmit. (T) (2) In LANs, a sequence of bits passed from one device to another along the transmission medium. When the token has data appended to it, it becomes a frame.

token ring. (1) According to IEEE 802.5, network technology that controls media access by passing a token (special packet or frame) between media-attached stations. (2) A FDDI or IEEE 802.5 network with a ring topology that passes tokens from one attaching ring station (node) to another.

trace. (1) A record of the execution of a computer program. It exhibits the sequences in which the instructions were executed. (A) (2) For data links, a record of the frames and bytes transmitted or received.

transaction program (TP). A program that processes transactions in an SNA network. There are two kinds of transaction programs: application transaction programs and service transaction programs. See also *conversation*.

transmission header (TH). Control information, optionally followed by a basic information unit (BIU) or a BIU segment, that is created and used by path control to route message units and to control their flow within the network.

trap. In the Simple Network Management Protocol (SNMP), a message sent by a managed node (agent function) to a management station to report an exception condition.

TRUE. Task-Related User Exit.

TSO. Time Sharing Option.

U

UA. Unnumbered acknowledgment.

UNBIND. In SNA, a request to deactivate a session between two logical units (LUs).

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

UTC. Coordinated universal time.

V

value. (1) A specific occurrence of an attribute; for example, “blue” for the attribute “color.” (T) (2) A quantity assigned to a constant, a variable, a parameter, or a symbol.

variable. (1) In programming languages, a language object that may take different values, one at a time. The values of a variable are usually restricted to a certain data type. (I) (2) A quantity that can assume any of a given set of values. (A) (3) A name used to represent a data item whose value can be changed while the program is running.

VGA. Video graphics adapter.

video graphics adapter (VGA). A computer adapter that provides high-resolution graphics and a total of 256 colors. See also *enhanced graphics adapter (EGA)*.

virtual machine (VM). (1) A virtual data processing system that appears to be at the exclusive disposal of a

particular user, but whose functions are accomplished by sharing the resources of a real data processing system. (T) (2) In VM/ESA, the virtual processors, virtual storage, virtual devices, and virtual channel subsystem allocated to a single user. A virtual machine also includes any expanded storage dedicated to it.

Virtual Machine/Conversational Monitoring System (VM/CMS). A time sharing system control program that manages the resources of an IBM System/370 or System/390 such that multiple remote terminal users have a functioning simulation of a computing system at their disposal. The Conversational Monitoring System provides general time sharing, development, and problem-solving facilities.

Virtual Telecommunications Access Method (VTAM). An IBM licensed program that controls communication and the flow of data in an SNA network. It provides single-domain, multiple-domain, and interconnected network capability.

VM. Virtual machine.

VM/CMS. Virtual Machine/Conversational Monitoring System.

VTAM. (1) Virtual Telecommunications Access Method. (2) Synonymous with *ACF/VTAM*.

W

window. (1) A portion of a display surface in which display images pertaining to a particular application can be presented. Different applications can be displayed simultaneously in different windows. (A) (2) An area with visible boundaries that presents a view of an object or with which a user conducts a dialog with a computer system. (3) In data communication, the number of data packets a data terminal equipment (DTE) or data circuit-terminating equipment (DCE) can send across a logical channel before waiting for authorization to send another data packet. The window is the main mechanism of pacing, or flow control, of packets.

window size. The specified number of frames of information that can be sent before receiving an acknowledgment response.

WinSock application programming interface (API). A socket-style transport interface developed for the Windows family of operating systems.

Index

Numerics

01, Connect Presentation Space 38
02, Disconnect Presentation Space 74
03, Send Key 82, 126, 151, 158
04, Wait 157
05, Copy Presentation Space 57
06, Search Presentation Space 121, 158
07, Query Cursor Location 98
08, Copy Presentation Space to String 62
09, Set Session Parameters 135
10, Query Sessions 105
11, Reserve 118
12, Release 117, 118
13, Copy OIA 47
14, Query Field Attribute 99
15, Copy String to Presentation Space 70
16/32 bit considerations 21
18, Pause 89, 156
20, Query System 107
21, Reset System 118, 119, 136
22, Query Session Status 42, 104
23, Start Host Notification 89, 101, 148
24, Query Host Update 90, 101, 156
25, Stop Host Notification 156
30, Search Field 119, 158
31, Find Field Position 44, 77
32-bit presentation space IDs 10
32, Find Field Length 44, 75
33, Copy String to Field 67
34, Copy Field to String 42
40, Set Cursor 135
41, Start Close Intercept 146
42, Query Close Intercept 95
43, Stop Close Intercept 155
50, Start Keystroke Intercept 151
51, Get Key 80, 94, 151
52, Post Intercept Status 94, 151
53, Stop Keystroke Intercept 157
90, Send File 123
91, Receive File 114
99, Convert Position or Convert RowCol 40
61, Lock PMSVC API 87
101, Connect Window Services 37
  Connect Window Services (101) 37
102, Disconnect Window Service 73
103, Query Window Coordinates 102
104, Window Status 91
105, Change Switch List LT Name 34
106, Change PS Window Name 32
110, Start Playing Macro 154
3270 terminal emulation 286

A

Allocate Communications Buffer (123) 31
applications
  error codes 311
  using SRPI 288
ASCII characters 17
ASCII Mnemonics 17
  general 17
  get key (51) function 19
  send key (3) function 19
ATTRB 138
attribute bytes 42, 57, 62, 67, 71, 138
AUTORESET 140

B

beep 94
BLANK 141
buffer size 289

C

C language
  init _ send _ req _ parms 290
  interface 289
  record definition 290
  requesters 289
  send _ request 290
  syntax 289
call (input) parameters
  general 29
call/return 284
calls, prerequisite 29
Change Menu Item 233, 381
Change PS Window Name (106) 32
Change Switch List LT Name (105) 34
character, escape 18, 81, 127, 139
characters, ASCII 17
command highlighting xxii
communication services functions
  Receive File (91) 114
  Send File (90) 123
Communications Manager
  applications 288
Compatibility with Personal Communications V4.1 with Windows 3.1 303
compile option highlighting xxiii
compiling and linking 9
Connect for Structured Fields (120) 35

© Copyright IBM Corp. 1989, 1997

Connect Presentation Space (1) 38, 126
functions where not required 39
interaction with disconnect 12
Connect Window Services (101) 37
Convert Position or Convert RowCol (99) 40
Copy Field to String (34) 24, 42
copy functions
Copy Field to String (34) 42
Copy OIA (13) 47
Copy Presentation Space (5) 57
Copy Presentation Space to String (8) 62
Copy String to Field (33) 67
Copy String to Presentation Space (15) 70
Copy OIA (13) 23, 47
Copy Presentation Space (5) 57
Copy Presentation Space to String (8) 23, 62
Copy String to Field (33) 24, 67
Copy String to Presentation Space (15) 70
CPRB (Connectivity Programming Request Block)
storage 289
create menu item 241, 387
critical sections 2
cross platform support 1
cursor movement 24

D

data structures 6
DDE data items
LU Topic 175
Session Topic 174
System Topic 174
DDE data items in a 16-bit environment 319
DDE data items, 16-bit
LU Topic 321
Session Topic 320
System Topic 320
DDE Functions in a 16-Bit Environment 319
Find Field 322
function list 321
Get Keystrokes 324
Get Mouse Input 325
Get Number of Close Requests 328
Get Operator Information Area 329
Get Partial Presentation Space 329
Get Presentation Space 332
Get Session Status 334
Get System Configuration 336
Get System Formats 337
Get System Status 338
Get System SysItems 339
Get System Topics 340
Get Trim Rectangle 341
Initiate Session Conversation 341
Initiate Structured Field Conversation 342
Initiate System Conversation 343
DDE Functions in a 16-Bit Environment *(continued)*
naming conventions for parameters 322
Put Data to Presentation Space 343
Search for String 344
Send Keystrokes 346
Session Execute Macro 347
Set Cursor Position 353
Set Mouse Intercept Condition 354
Set Presentation Space Service Condition 356
Set Session Advise Condition 358
Set Structured Field Service Condition 359
Start Close Intercept 360
Start Keystroke Intercept 361
Start Mouse Input Intercept 363
Start Read SF 366
Start Session Advise 367
Stop Close Intercept 369
Stop Keystroke Intercept 369
Stop Mouse Input Intercept 370
Stop Read SF 371
Stop Session Advise 371
summary of DDE functions in a 16-bit environment 374
Terminate Session Conversation 372
Terminate Structured Field Conversation 372
Terminate System Conversation 373
Write SF 373
DDE functions in a 32-bit environment 173
DDE data items, 32-bit 173
LU Topic 175
Session Topic 174
System Topic 174
Find Field 176
function list 175
Get Keystrokes 178
Get Mouse Input 179
Get Number of Close Requests 182
Get Operator Information Area 183
Get Partial Presentation Space 184
Get Presentation Space 186
Get Session Status 189
Get System Configuration 191
Get System Formats 192
Get System Status 192
Get System SysItems 193
Get System Topics 194
Get Trim Rectangle 195
Initiate Session Conversation 196
Initiate Structured Field Conversation 197
Initiate System Conversation 197
naming conventions for parameters 176
Put Data to Presentation Space 198
Search for String 199
Send Keystrokes 200
Session Execute Macro 201
Set Cursor Position 208

DDE functions in a 32-bit environment *(continued)*
Set Mouse Intercept Condition 210
Set Presentation Space Service Condition 212
Set Session Advise Condition 214
Set Structured Field Service Condition 216
Start Close Intercept 217
Start Keystroke Intercept 218
Start Mouse Input Intercept 219
Start Read SF 222
Start Session Advise 224
Stop Close Intercept 225
Stop Keystroke Intercept 226
Stop Mouse Input Intercept 227
Stop Read SF 227
Stop Session Advise 228
Terminate Session Conversation 229
Terminate Structured Field Conversation 229
Terminate System Conversation 229
Write SF 230
DDE menu functions in a 16-bit environment 378
Change Menu Item 381
create menu item 387
Initiate Menu Conversation 388
list 380
Start Menu Advise 389
Stop Menu Advise 390
Terminate Menu Conversation 391
DDE menu functions in a 32-bit environment 231
Change Menu Item 233
create menu item 241
Initiate Menu Conversation 242
list 233
Start Menu Advise 242
Stop Menu Advise 244
Terminate Menu Conversation 245
debugging 19
default
values 288
default directory
Receive File 117
device services functions
Get Key (51) 80
Post Intercept Status (52) 94
Release (12) 117
Reserve (11) 118
Start Keystroke Intercept (50) 151
Stop Keystroke Intercept (53) 157
directory, default
Receive File 117
Send File 126
Disconnect from Structured Fields (121) 72
Disconnect Presentation Space (2) 74
interaction with connect 12
Disconnect Window Service (102) 73
dynamic link method 10

E

EAB 141
EHLLAPI
function names xxii
functions 29
summary 29
EHLLAPI call format 6
EHLLAPI programming overview 5
EHLLAPI return codes 7
EOT 137
error handling 311
ESC 139
escape character 18, 81, 127, 139
exception code values 315
exception object values 316

F

field-formatted PS 44, 120
field-related functions 99
Copy Field to String (34) 42
Copy String to Field (33) 67
Find Field Length (32) 75
Find Field Position (31) 77
Search Field (30) 119
fields, host
input protected 126
numeric only 126
file transfer functions
Receive File (91) 114
Send File (90) 123
Find Field 176, 322
Find Field Length (32) 44, 75
Find Field Position (31) 44, 77
flow, requester and server 286
FPAUSE 138
Free Communications Buffer (124) 79
function calls
call (input) parameters 29
notes on using the function 29
page layout conventions 29
prerequisite calls 29
return (output) parameters 29
use of 29
function names xxii
function number parameter xxii

G

Get Key (51) 17, 80, 94, 151
Get Keystrokes 178, 324
Get Mouse Input 179, 325
Get Number of Close Requests 182, 328
Get Operator Information Area 183, 329

Get Partial Presentation Space 184, 329
Get Presentation Space 186, 332
Get Request Completion (125) 83
Get Session Status 189, 334
Get System Configuration 191, 336
Get System Formats 192, 337
Get System Status 192, 338
Get System SysItems 193, 339
Get System Topics 194, 340
Get Trim Rectangle 195, 341

H

highlighting xxiii
  ASCII mnemonics xxiii
  commands
    compile xxiii
    link xxiii
  displayed text xxiii
  entry-point names xxiii
  function names xxiii
  keywords xxiii
  operator input xxiii
  options
    compile xxiii
    link xxiii
    session xxiii
  programmer input xxiii
  sample code xxiii
  variable names xxiii
host
  computer router 285
  computer server 285
host automation scenarios 22
host fields
  input protected 126
  numeric only 126
host-connected presentation space 11

I

IBM Support Center 107
init _ send _ req _ parms
  C language 290
initialization 9
Initiate Menu Conversation 242, 388
Initiate Session Conversation 196, 341
Initiate Structured Field Conversation 197, 342
Initiate System Conversation 197, 343
input protected fields 126
introduction to EHLLAPI programming 5
introduction to Emulator APIs
  Dynamic Data Exchange (DDE) 1
  Emulator High Level Language API (EHLLAPI) 1
  Personal Communications Session API (PCSAPI) 1
  Server-Requestor Programming Interface (SRPI) 1
invoking SEND _ REQUEST 289
IPAUSE 138

K

keyboard mnemonics
  See also “mnemonics.” 17
keyboard, session 17
keystroke interception, Get Key (51) 80
keyword highlighting xxii

L

language interface
  C language 289
languages 5
licensing agreement 447
link option highlighting xxiii
Lock Presentation Space API (60) 85
Lock Window Services API (61) 87
locking presentation space 16
LWAIT 140, 158

M

memory allocation 7
mnemonics 71
  for Send Key 17
mnemonics, ASCII 17
mnemonics, shift key 17

N

NOATTRB 138
NOBLANK 141
NOEAB 141
NOQUIET 138
NORESET 140
NOXLATE 141
NULLATTRB 138
numeric only fields 126
NWAIT 140, 158

O

OIA 47, 158
Operator Information Area
  See “OIA.” 47
operator services functions
  Pause (18) 89
  Query Host Update (24) 101
  Query Session Status (22) 104
  Query Sessions (10) 105
  Query System (20) 107
  Reset System (21) 119
  Send Key (3) 126
  Set Session Parameters (9) 135

operator services functions *(continued)*
Start Host Notification (23) 148
Stop Host Notification (25) 156
Wait (4) 157
options 137, 145
options, session, highlighting xxiii

P

parameters
call 29
function number xxii
returned 288
SEND _ REQUEST 286
supplied 287
path, default
Receive File 117
Send File 126
Pause (18) 23, 89, 156
PCSAPI 165
how to use 165
pcsConnectSession 166
pcsDisconnectSession 166
pcsQueryEmulatorStatus 167
pcsQuerySessionList 167
pcsQueryWorkstationProfile 169
pcsStartSession 170
pcsStopSession 170
pcsDisconnectSession 166
pcsQueryEmulatorStatus 167
pcsQuerySessionList 167
pcsQueryWorkstationProfile 169
pcsStartSession 170
pcsStopSession 170
performance considerations 289
Post Intercept Status (52) 27, 94, 151
prerequisite calls
general 29
presentation services functions
Connect Presentation Space (1) 38
Copy Field to String (34) 42
Copy OIA (13) 47
Copy Presentation Space (5) 57
Copy Presentation Space to String (8) 62
Copy String to Field (33) 67
Copy String to Presentation Space (15) 70
Disconnect Presentation Space (2) 74
Find Field Length (32) 75
Find Field Position (31) 77
Get Request Completion (125) 83
Lock Presentation space API (60) 85
Query Cursor Location (7) 98
Query Field Attribute (14) 99
Search Field (30) 119
Search Presentation Space (6) 121
Set Cursor (40) 135
presentation space 47
character table 49
cursor movement 24
field-formatted 42, 44, 67, 75, 77, 119, 120
host-connected 11
maximum number of 12
how specified 11
identifier
blank specifier 13
function 11
how processed 12
letter specifier 13
null specifier 13
processing for functions not requiring connect 13
processing for functions requiring connect 12
presentation space names 11
declaring 11
maximum number of 11
valid names 11
presentation spaces 10
programming interface, server-requester 284
PSID handling
functions not requiring connect 13
functions requiring connect 12
Put Data to Presentation Space 198, 343

Q

Query Close Intercept (42) 95
Query Communications Buffer Size (122) 96
Query Cursor Location (7) 98
Query Field Attribute (14) 99
Query Host Update (24) 90, 101, 148, 156
Query Session Status (22) 42, 104
Query Sessions (10) 105
Query System (20) 107
Query Window Coordinates (103) 102
QUIET 138

R

Read Structured Fields (126) 109
Receive File (91) 24, 114, 115, 139
default path for target file 117
RECEIVE.EXE location 115
record definition
C language 290
related publications xxiii
relationship requester server 285
Release (12) 25, 117, 118
requester
C language 289
server flow, and 286
server relationship 285
Reserve (11) 25, 117, 118

Reset System (21) 117, 118, 119, 136
return (output) parameters
  general 29
return codes 311
REXX EHLLAPI functions
  Send_File 432
  Sendkey 433
  Set_Cursor_Pos 434
  Set_Session_Parms 435
  Set_Window_Status 436
  Start_Close_Intercept 437
  Start_Host_Notify 438
  Start_Keystroke_Intercept 439
  Stop_Close_Intercept 440
  Stop_Host_Notify 441
  Stop_Keystroke_Intercept 442
  Wait 443
REXX EHLLAPI programming notes 443
router
  SRPI 285, 286

S

sample code highlighting xxiii
sample program, a simple EHLLAPI 19
sample programs 3
Scenario 1. A Search Function 23
Scenario 2. Sending Keystrokes 24
Scenario 4. File Transfer 25
Scenario 5. Automation 25
Scenario 6. Keystroke Filtering 25
Scenario 7. Keyboard Enhancement 26
Search Field (30) 119, 158
Search for String 199, 344
search functions
  Search Field (30) 119
  Search Presentation Space (6) 121
Search Presentation Space (6) 23, 121, 158
SEND _ REQUEST
  invoking 289
  parameters
    returned 288
    supplied 287
  processing errors 311
  routing 285
send _ request function
  C language 290
Send File (90) 24, 123, 139
  default path for target file 126
  SEND.EXE location 124
Send Key (3) 17, 82, 126, 151, 158
Send Keystrokes 200, 346
Send_File 432
sending keystrokes
  mnemonics 17
  Send Key (3) 126
Sendkey 433
server
  name 288
  return codes 317
server-requester programming interface 284
service 107
Session Execute Macro 201, 347
session keyboard 17
session options highlighting xxiii
Set Cursor (40) 135
Set Cursor Position 208, 353
Set Mouse Intercept Condition 210, 354
Set Presentation Space Service Condition 212, 356
Set Session Advise Condition 214, 358
Set Session Parameters (9) 127
  general 135
  List of affected functions 136
  Valid Input 137, 145
Set Structured Field Service Condition 216, 359
Set_Cursor_Pos 434
Set_Session_Parms 435
Set_Window_Status 436
shift key mnemonics 17
size of presentation spaces 11
source code syntax 22
specifying strings 71
SRCHALL 137
SRCHBKWD 138
SRCHFROM 137
SRCHFRWD 138
stack size 3
Start Close Intercept 217, 360
Start Close Intercept (41) 146
Start Host Notification (23) 89, 101, 138, 148, 156
Start Keystroke Intercept 218, 361
Start Keystroke Intercept (50) 151
Start Menu Advise 242, 389
Start Mouse Input Intercept 219, 363
Start Playing Macro (110) 154
Start Read SF 222, 366
Start Session Advise 224, 367
Start_Close_Intercept 437
Start_Host_Notify 438
Start_Keystroke_Intercept 439
static link method 9
Stop Close Intercept 225, 369
Stop Close Intercept (43) 155
Stop Host Notification (25) 156
Stop Keystroke Intercept 226, 369
Stop Keystroke Intercept (53) 157
Stop Keystroke Intercept (53) you can call the 27
Stop Menu Advise 244, 390
Stop Mouse Input Intercept 227, 370
Stop Read SF 227, 371
Stop Session Advise 228, 371

Stop_Close_Intercept 440
Stop_Host_Notify 441
Stop_Keystroke_Intercept 442
STREOT 137
string interception, Get Key (51) 80
string specification 137
  Send File 126
STRLEN 137
syntax
  C language 289

T

Terminate Menu Conversation 245, 391
Terminate Session Conversation 229, 372
Terminate Structured Field Conversation 229, 372
Terminate System Conversation 229, 373
TIMEOUT 139
trademarks 449
transport layer errors 311
TWAIT 140, 158
types of presentation spaces 11

U

UERCPRB
  C language 290
using API header files 2

V

variable names, highlighting xxiii

W

Wait 443
Wait (4) 23, 127, 157
window services functions
  Change PS Window Name (106) 32
  Change Switch List LT Name (105) 34
  Lock Window Services API (61) 87
Window Status (104) 91
Write SF 230, 373
Write Structured Fields (127) 158

X

XLATE 141

Communicating Your Comments to IBM

eNetwork Personal Communications
Version 4.2
for Windows 95 and Windows NT
Emulator Programming

Publication No. SC31-8478-00

If you especially like or dislike anything about this book, please use one of the methods listed below to send your comments to IBM. Whichever method you choose, make sure you send your name, address, and telephone number if you would like a reply.

Feel free to comment on specific errors or omissions, accuracy, organization, subject matter, or completeness of this book. However, the comments you send should pertain to only the information in this manual and the way in which the information is presented. To request additional publications, or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer.

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

If you are mailing a readers' comment form (RCF) from a country other than the United States, you can give the RCF to the local IBM branch office or IBM representative for postage-paid mailing.

If you prefer to send comments by mail, use the RCF at the back of this book.

If you prefer to send comments by FAX, use this number:

United States and Canada: **1-800-227-5088**

If you prefer to send comments electronically, use this network ID:

- IBM Mail Exchange: **USIB2HPD at IBMMAIL**
- IBMLink: **CIBMORCF at RALVM13**
- Internet: **USIB2HPD@VNET.IBM.COM**

Make sure to include the following in your note:

Title and publication number of this book
Page number or topic to which your comment applies.

Help us help you!

eNetwork Personal Communications
Version 4.2
for Windows 95 and Windows NT
Emulator Programming

Publication No. SC31-8478-00

If your concern is service related, you can reach Service at 1-800-992-4777 in the United States. Outside the United States, please check your phone listing for the IBM Service Center nearest you.

We hope you find this publication useful, readable and technically accurate, but only you can tell us! Please take a few minutes to let us know what you think by completing this form.

Overall, how satisfied are you with the information in this book?	Satisfied	Dissatisfied

How satisfied are you that the information in this book is:	Satisfied	Dissatisfied
Accurate		
Complete		
Easy to find		
Easy to understand		
Well organized		
Applicable to your task		

Specific Comments or Problems:

Please tell us how we can improve this book:

Thank you for your response. When you send information to IBM, you grant IBM the right to use or distribute the information without incurring any obligation to you. You of course retain the right to use the information in any way you choose.

Your Internet Address: _____

Name

Address

Company or Organization

Phone No.

Help us help you!
SC31-8478-00

Cut or Fold
Along Line

Fold and Tape **Please do not staple** Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

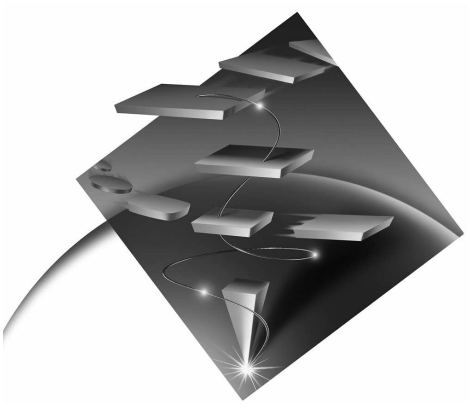
Information Development
Department CGMD
International Business Machines Corporation
PO BOX 12195
RESEARCH TRIANGLE PARK NC 27709-9990

Fold and Tape **Please do not staple** Fold and Tape

SC31-8478-00

Cut or Fold
Along Line

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



SC31-8478-00

Spine information:

eNetwork Personal Communications
Version 4.2
for Windows 95 and Windows NT

Emulator Programming