

Personal Communications for Windows, Version 6.0



Emulator Programming

Personal Communications for Windows, Version 6.0



Emulator Programming

Note

Before using this information and the product it supports, read the information in Appendix G, "Notices," on page 485.

Eleventh Edition (September 2009)

This edition applies to Version 6.0 of IBM Personal Communications for Windows (program number: 5639-I70) and to all subsequent releases and modifications until otherwise indicated in new editions.

© **Copyright International Business Machines Corporation 1989, 2009.**

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Figures ix

Tables xi

About This Book xiii

Who Should Read This Book xiii
Where To Find More Information xiii
Support Options xiv
 Subscribing to Support News xiv
 Support Assistant xiv
Notation xv

Chapter 1. Introduction to Emulator APIs 1

Using API Header Files 2
Critical Sections 2
Stack Size 2
Running 16-bit Windows EHLLAPI programs 2
Windows x64 Platform Support 2
Sample Programs 3
 Displaying Arabic data in the VBHLLAPI sample
 program 4

**Chapter 2. Introduction to IBM Standard
EHLLAPI, IBM Enhanced EHLLAPI and
WinHLLAPI Programming 5**

EHLLAPI Overviews 5
 IBM Standard EHLLAPI 5
 WinHLLAPI 5
 WinHLLAPI and IBM Standard EHLLAPI 5
 IBM Enhanced EHLLAPI and IBM Standard
 EHLLAPI 6
Languages 6
EHLLAPI Call Format 6
Data Structures 7
Memory Allocation 8
EHLLAPI Return Codes 8
Compiling and Linking 9
 Static Link Method 10
 Dynamic Link Method 10
 Multithreading 11
Presentation Spaces 11
 IBM Enhanced 32-Bit Interface Presentation Space
 IDs 11
 Types of Presentation Spaces 11
 Size of Presentation Spaces 11
 Presentation Space IDs 12
 Host-Connected Presentation Space 12
 Presentation Space ID Handling 12
 Sharing EHLLAPI Presentation Space between
 Processes 14
 Using mouse actions to select, copy, and paste
 text in the Presentation Space 16
 ASCII Mnemonics 17
 Debugging 19

A Simple EHLLAPI Sample Program 19
Standard and Enhanced Interface Considerations 21
Host Automation Scenarios 22

Chapter 3. EHLLAPI Functions 27

Unicode Support for Code Pages 1390/1399 and
1137 27
Page Layout Conventions 27
 Prerequisite Calls 28
 Call Parameters 28
 Return Parameters 28
 Notes on Using This Function 28
Summary of EHLLAPI Functions 28
 Allocate Communications Buffer (123) 30
 Cancel File Transfer (92) 31
 Change PS Window Name (106) 32
 Change Switch List LT Name (105) 33
 Connect for Structured Fields (120) 34
 Connect Presentation Space (1) 36
 Connect Window Services (101) 37
 Convert Position or Convert RowCol (99) 38
 Copy Field to String (34) 40
 Copy OIA (13) 48
 Copy Presentation Space (5) 57
 Copy Presentation Space to String (8) 64
 Copy String to Field (33) 72
 Copy String to Presentation Space (15) 76
 Disconnect from Structured Fields (121) 80
 Disconnect Presentation Space (2) 82
 Disconnect Window Service (102) 82
 EditKey Intercept 83
 Find Field Length (32) 85
 Find Field Position (31) 86
 Free Communications Buffer (124) 88
 Get Key (51) 89
 Get Request Completion (125) 94
 Lock Presentation Space API (60) 97
 Lock Window Services API (61) 99
 Pause (18) 100
 Post Intercept Status (52) 102
 Query Additional Field Attribute (45) 103
 Query Close Intercept (42) 104
 Query Communications Buffer Size (122) 104
 Query Communication Event (81) 106
 Query Cursor Location (7) 107
 Query Field Attribute (14) 107
 Query Host Update (24) 109
 Query Session Status (22) 110
 Query Sessions (10) 112
 Query System (20) 113
 Query Window Coordinates (103) 115
 Read Structured Fields (126) 116
 Receive File (91) 121
 Release (12) 123
 Reserve (11) 123

| | |
|---|-----|
| Reset System (21) | 124 |
| Search Field (30) | 125 |
| Search Presentation Space (6) | 129 |
| Send File (90) | 133 |
| Send Key (3) | 135 |
| Set Cursor (40) | 146 |
| Set Session Parameters (9) | 147 |
| Start Close Intercept (41) | 156 |
| Start Communication Notification (80) | 158 |
| Start Host Notification (23) | 160 |
| Start Keystroke Intercept (50) | 163 |
| Start Playing Macro (110) | 165 |
| Stop Close Intercept (43) | 166 |
| Stop Communication Notification (82) | 167 |
| Stop Host Notification (25) | 168 |
| Stop Keystroke Intercept (53) | 168 |
| Wait (4) | 169 |
| Window Status (104) | 170 |
| Write Structured Fields (127) | 173 |

Chapter 4. WinHLLAPI Extension

Functions 179

| | |
|--|-----|
| Summary of WinHLLAPI Functions | 179 |
| WinHLLAPI Asynchronous Functions | 179 |
| WinHLLAPIAsync | 179 |
| WinHLLAPICancelAsyncRequest | 185 |
| Initialization and Termination Functions | 186 |
| WinHLLAPIStartup | 186 |
| WinHLLAPICleanup | 187 |
| Blocking Routines | 187 |
| WinHLLAPIsBlocking | 187 |
| WinHLLAPISetBlockingHook | 187 |
| WinHLLAPIUnhookBlockingHook | 188 |
| WinHLLAPICancelBlockingCall | 188 |

Chapter 5. PCSAPI Functions 191

| | |
|--|-----|
| How to Use PCSAPI | 191 |
| Page Layout Conventions | 191 |
| Function Type | 191 |
| Parameter Type and Description | 191 |
| Return Code | 191 |
| pcsConnectSession | 191 |
| Function Type | 192 |
| Parameter Type and Description | 192 |
| Return Code | 192 |
| pcsDisconnectSession | 192 |
| Function Type | 192 |
| Parameter Type and Description | 192 |
| Return Code | 192 |
| pcsQueryConnectionInfo | 193 |
| Function Type | 193 |
| Parameter Type and Description | 193 |
| Return Code | 193 |
| ConnectionInfo | 193 |
| Example | 193 |
| pcsQueryEmulatorStatus | 194 |
| Function Type | 194 |
| Parameter Type and Description | 194 |
| Return Code | 194 |
| pcsQuerySessionList | 194 |

| | |
|--|-----|
| Function Type | 195 |
| Parameter Type and Description | 195 |
| Return Parameters | 195 |
| Example | 196 |
| pcsQueryWorkstationProfile | 196 |
| Function Type | 196 |
| Parameter Type and Description | 196 |
| Return Code | 196 |
| pcsSetLinkTimeout | 197 |
| Function Prototype | 197 |
| Parameter Type and Description | 197 |
| Return Code | 197 |
| pcsStartSession | 197 |
| Function Type | 197 |
| Parameter Type and Description | 197 |
| Return Code | 198 |
| pcsStopSession | 198 |
| Function Type | 198 |
| Parameter Type and Description | 198 |
| Return Code | 198 |
| Page Setup Functions | 199 |
| Restrictions | 199 |
| pcsGetPageSettings | 199 |
| pcsRestorePageDefaults | 201 |
| pcsSetPageSettings | 202 |
| Printer Setup Functions | 205 |
| Restrictions | 205 |
| pcsGetPrinterSettings | 205 |
| pcsSetPrinterSettings | 210 |

Chapter 6. DDE Functions in a 32-bit Environment 215

| | |
|---|-----|
| Personal Communications DDE Data Items | 215 |
| Using System Topic Data Items | 216 |
| Using Session Topic Data Items | 216 |
| Using LU Topic Data Items (3270 Only) | 216 |
| DDE Functions | 216 |
| Naming Conventions for Parameters | 217 |
| Code Conversion | 218 |
| Conversion Types | 218 |
| Personal Communications Response | 219 |
| Find Field | 220 |
| CF_DSPTTEXT | 220 |
| CF_TEXT | 220 |
| Personal Communications Response | 221 |
| Structure of the Field Information | 221 |
| Get Keystrokes | 222 |
| Personal Communications Response | 223 |
| Structure of the Keystroke Information | 223 |
| Get Mouse Input | 223 |
| Personal Communications Response | 224 |
| Structure of the Mouse Input Information | 224 |
| Get Number of Close Requests | 226 |
| Personal Communications Response | 226 |
| Structure of the Number of the Close Requests Information | 227 |
| Get Operator Information Area | 227 |
| Personal Communications Response | 227 |
| Structure of the Operator Information Area | 228 |
| Get Partial Presentation Space | 228 |
| Personal Communications Response | 228 |

| | | | |
|--|-----|---|-----|
| Structure of the Presentation Space | 229 | Start Session Advise | 266 |
| Get Presentation Space | 230 | Personal Communications Response. | 267 |
| Personal Communications Response. | 231 | Stop Close Intercept | 267 |
| Structure of the Presentation Space | 231 | Personal Communications Response. | 267 |
| Get Session Status | 232 | Stop Keystroke Intercept. | 268 |
| Personal Communications Response. | 233 | Personal Communications Response. | 268 |
| Format of Status Information | 233 | Stop Mouse Input Intercept. | 268 |
| Get System Configuration | 234 | Personal Communications Response. | 269 |
| Personal Communications Response. | 234 | Stop Read SF | 269 |
| Format of System Configuration Information | 235 | PC/3270 response | 269 |
| Get System Formats | 235 | Stop Session Advise | 270 |
| Personal Communications Response. | 235 | Personal Communications Response. | 270 |
| Get System Status | 236 | Terminate Session Conversation | 270 |
| Personal Communications Response. | 236 | Personal Communications Response. | 271 |
| Get System SysItems | 237 | Terminate Structured Field Conversation | 271 |
| Personal Communications Response. | 237 | PC/3270 Response | 271 |
| Get System Topics | 238 | Terminate System Conversation | 271 |
| Personal Communications Response. | 238 | Personal Communications Response. | 271 |
| Get Trim Rectangle | 238 | Write SF | 272 |
| Personal Communications Response. | 239 | PC/3270 Response | 272 |
| Initiate Session Conversation | 239 | DDE Menu Item API in a Windows 32-Bit | |
| Personal Communications Response. | 240 | Environment | 273 |
| Initiate Structured Field Conversation | 240 | DDE Menu Client | 273 |
| PC/3270 Response | 240 | DDE Menu Server. | 273 |
| Initiate System Conversation | 240 | DDE Menu Functions | 274 |
| Personal Communications Response. | 241 | Change Menu Item | 275 |
| Put Data to Presentation Space | 241 | Create Menu Item | 281 |
| Personal Communications Response. | 242 | Initiate Menu Conversation. | 282 |
| Search for String | 242 | Start Menu Advise | 282 |
| Personal Communications Response. | 243 | Stop Menu Advise. | 284 |
| Structure of the Search Information | 243 | Terminate Menu Conversation. | 284 |
| Send Keystrokes | 243 | Summary of DDE Functions in a Windows | |
| Personal Communications Response. | 244 | 32-Bit Environment | 285 |
| Session Execute Macro | 244 | | |
| Personal Communications Response. | 245 | | |
| Issuing Commands with the Session Execute | | | |
| Macro Function. | 245 | | |
| WINDOW Command | 245 | | |
| KEYBOARD Command | 245 | | |
| SEND Command | 246 | | |
| RECEIVE Command | 246 | | |
| SENDKEY Command | 246 | | |
| WAIT Command | 250 | | |
| Set Cursor Position | 251 | | |
| Personal Communications Response. | 252 | | |
| Set Mouse Intercept Condition | 253 | | |
| Personal Communications Response. | 255 | | |
| Set Presentation Space Service Condition | 255 | | |
| Personal Communications Response. | 257 | | |
| Set Session Advise Condition | 257 | | |
| Personal Communications Response. | 258 | | |
| Set Structured Field Service Condition | 258 | | |
| PC/3270 Response | 259 | | |
| Start Close Intercept | 259 | | |
| Personal Communications Response. | 260 | | |
| Start Keystroke Intercept | 260 | | |
| Personal Communications Response. | 261 | | |
| Start Mouse Input Intercept | 261 | | |
| Personal Communications Response. | 262 | | |
| Start Read SF | 264 | | |
| PC/3270 Response | 265 | | |

Chapter 7. Using DDE Functions with a DDE Client Application 291

| | |
|--|-----|
| Using the Personal Communications DDE Interface | 291 |
| System Conversation | 292 |
| Session Conversation | 292 |
| Session Conversation (Hot Link) | 294 |
| Personal Communications DDE Interface | 295 |
| DDE Functions for System Conversation | 295 |
| Get System Configuration | 295 |
| Get System Formats | 296 |
| Get System Status | 296 |
| Get System SysItems | 297 |
| Get System Topics | 297 |
| Initiate System Conversation | 297 |
| Terminate System Conversation | 298 |
| DDE Functions for Session Conversation | 298 |
| Find Field | 298 |
| Get Operator Information Area | 299 |
| Get Partial Presentation Space | 300 |
| Get Presentation Space | 301 |
| Get Session Status | 302 |
| Get Trim Rectangle | 302 |
| Initiate Session Conversation | 304 |
| Put Data to Presentation Space | 305 |
| Search for String | 305 |
| Session Execute Macro | 306 |
| Set Cursor Position | 307 |

| | |
|---|-----|
| Terminate Session Conversation | 307 |
| DDE Functions for Session Conversation (Hot Link) | 307 |
| Initiate Session Conversation | 307 |
| Start Close Intercept | 308 |
| Start Keystroke Intercept | 308 |
| Start Session Advise | 309 |
| Stop Close Intercept | 310 |
| Stop Keystroke Intercept. | 311 |
| Stop Session Advise | 311 |
| Terminate Session Conversation | 311 |
| Visual Basic Sample Program | 311 |

Chapter 8. Server-Requester Programming Interface (SRPI) Support 323

| | |
|---|-----|
| How to Use SRPI | 323 |
| SRPI Compatibility | 323 |
| Using the Server-Requester Programming Interface | 324 |
| SEND_REQUEST Parameters | 326 |
| Supplied Parameters | 326 |
| Returned Parameters | 328 |
| How PC/3270 Applications Use SRPI | 328 |
| Invoking SEND_REQUEST | 329 |
| Performance Considerations | 329 |
| Handling the Interrupt (Ctrl+Break) Key | 329 |
| C Requesters | 329 |
| C send_request Function | 330 |
| SRPI Record Definition | 330 |
| SRPI Return Codes | 330 |

Appendix A. Query Reply Data Structures Supported by EHLLAPI . . . 331

| | |
|---|-----|
| The DDM Query Reply | 331 |
| DDM Application Name Self-Defining Parameter | 332 |
| PCLK Protocol Controls Self-Defining Parameter | 332 |
| Base DDM Query Reply Formats | 332 |
| The IBM Auxiliary Device Query Reply | 334 |
| Optional Parameters | 334 |
| Direct Access Self-Defining Parameter | 335 |
| PCLK Protocol Controls Self-Defining Parameter | 336 |
| The OEM Auxiliary Device Query Reply | 336 |
| Direct Access Self-Defining Parameter | 337 |
| PCLK Protocol Controls Self-Defining Parameter | 337 |
| The Cooperative Processing Requester Query Reply | 338 |
| The Product-Defined Query Reply | 338 |
| Optional Parameters | 338 |
| Direct Access Self-Defining Parameter | 339 |
| The Document Interchange Architecture Query Reply | 340 |

Appendix B. Differences from Communication Manager/2 EHLLAPI . 343

| | |
|-------------------------------------|-----|
| Set Session Parameter (9) | 343 |
| Set Options | 343 |
| Return Parameters. | 343 |
| EAB Option | 343 |
| Copy OIA (13) | 344 |
| Copy String to PS (15) | 344 |
| Storage Manager (17). | 345 |

| | |
|--|-----|
| Copy String to Field (33) | 345 |
| Get Key (51). | 345 |
| Window Status (104) | 345 |
| Query Sessions (10) | 345 |
| Connect for Structured Fields (120) | 345 |
| Allocate Communications Buffer (123) | 345 |
| ASCII Mnemonics | 346 |
| Get Request Completion (125) | 346 |

Appendix C. DOS-Mode EHLLAPI for Windows 347

| | |
|------------------------|-----|
| Installation | 347 |
|------------------------|-----|

Appendix D. SRPI Return Codes . . . 349

| | |
|---|-----|
| Error Handling | 349 |
| Transport Layer Errors | 349 |
| Application Errors. | 349 |
| SEND_REQUEST Processing Errors | 349 |
| Types of SRPI Return Codes | 349 |
| Type 0 Return Code Definitions | 350 |
| Type 1 Return Code Definitions | 350 |
| Type 2 Return Code Definitions | 352 |
| Type 3 Return Code Definitions | 353 |
| Class Definitions for Type 2 and Type 3 | 353 |
| Exception Code Values for Type 2 and Type 3 | 354 |
| Exception Object Values for Type 2 and Type 3 | 354 |
| Server Return Codes | 355 |

Appendix E. DDE Functions in a 16-Bit Environment 357

| | |
|--|-----|
| Personal Communications DDE Data Items in a 16-Bit Environment | 357 |
| Using System Topic Data Items | 358 |
| Using Session Topic Data Items | 358 |
| Using LU Topic Data Items (PC/3270 Only) | 358 |
| DDE Functions in a 16-Bit Environment | 358 |
| Naming Conventions for Parameters | 359 |
| Find Field | 360 |
| Get Keystrokes | 361 |
| Get Mouse Input | 362 |
| Get Number of Close Requests | 365 |
| Get Operator Information Area | 366 |
| Get Partial Presentation Space | 367 |
| Get Presentation Space | 369 |
| Get Session Status | 371 |
| Get System Configuration | 372 |
| Get System Formats | 373 |
| Get System Status | 374 |
| Get System SysItems | 375 |
| Get System Topics | 376 |
| Get Trim Rectangle | 377 |
| Initiate Session Conversation | 377 |
| Initiate Structured Field Conversation | 378 |
| Initiate System Conversation | 379 |
| Put Data to Presentation Space | 379 |
| Search for String | 380 |
| Send Keystrokes | 381 |
| Session Execute Macro | 382 |
| Set Cursor Position | 388 |
| Set Mouse Intercept Condition | 390 |

| | |
|--|-----|
| Set Presentation Space Service Condition | 392 |
| Set Session Advise Condition | 393 |
| Set Structured Field Service Condition | 394 |
| Start Close Intercept | 395 |
| Start Keystroke Intercept | 397 |
| Start Mouse Input Intercept | 398 |
| Start Read SF | 401 |
| Start Session Advise | 402 |
| Stop Close Intercept | 403 |
| Stop Keystroke Intercept. | 404 |
| Stop Mouse Input Intercept. | 405 |
| Stop Read SF | 405 |
| Stop Session Advise | 406 |
| Terminate Session Conversation | 407 |
| Terminate Structured Field Conversation | 407 |
| Terminate System Conversation | 407 |
| Write SF | 408 |
| DDE Menu Item API in a 16-Bit Environment | 409 |
| DDE Menu Client in a 16-Bit Environment | 409 |
| DDE Menu Server, 32-Bit | 410 |
| DDE Menu Functions in a 16-Bit Environment | 411 |
| Change Menu Item | 411 |
| Create Menu Item | 416 |
| Initiate Menu Conversation. | 417 |
| Start Menu Advise | 418 |
| Stop Menu Advise. | 419 |
| Terminate Menu Conversation. | 420 |
| Summary of DDE Functions in a 16-Bit Environment | 420 |

Appendix F. REXX EHLLAPI Functions 425

| | |
|---|-----|
| Overview of REXX EHLLAPI Function Calls and Return Values | 425 |
| Installation | 425 |
| Conventions. | 425 |
| Summary of Prerequisite Calls for Functions | 426 |
| Summary of EHLLAPI and REXX EHLLAPI Functions. | 427 |
| Change_Switch_Name | 430 |
| Change_Window_Name. | 431 |
| Connect | 432 |
| Connect_PM. | 433 |
| Convert_Pos. | 434 |
| Copy_Field_To_Str | 435 |
| Copy_OIA | 436 |
| Copy_PS | 437 |
| Copy_PS_To_Str | 438 |
| Copy_Str_To_Field | 439 |
| Copy_Str_To_PS | 440 |
| Disconnect | 441 |

| | |
|-------------------------------------|-----|
| Disconnect_PM. | 442 |
| Find_Field_Len. | 443 |
| Find_Field_Pos. | 444 |
| Get_Key | 445 |
| Get_Window_Status | 446 |
| Intercept_Status | 447 |
| Lock_PMSVC | 448 |
| Lock_PS | 449 |
| Pause | 450 |
| Query_Close_Intercept | 451 |
| Query_Cursor_Pos | 452 |
| Query_Emulator_Status | 453 |
| Query_Field_Attr | 454 |
| Query_Host_Update | 455 |
| Query_Session_List | 456 |
| Query_Session_Status | 457 |
| Query_Sessions. | 458 |
| Query_System | 459 |
| Query_Window_Coord | 460 |
| Query_Workstation_Profile | 461 |
| Receive_File | 462 |
| Release | 463 |
| Reserve | 464 |
| Reset_System | 465 |
| Search_Field. | 466 |
| Search_PS | 467 |
| Send_File. | 468 |
| Sendkey | 469 |
| Set_Cursor_Pos. | 470 |
| Set_Session_Parms. | 471 |
| Set_Window_Status | 472 |
| Start_Close_Intercept | 473 |
| Start_Communication | 474 |
| Start_Host_Notify | 475 |
| Start_Keystroke_Intercept | 476 |
| Start_Session | 477 |
| Stop_Close_Intercept | 478 |
| Stop_Communication. | 479 |
| Stop_Host_Notify | 480 |
| Stop_Keystroke_Intercept | 481 |
| Stop_Session | 482 |
| Wait | 483 |
| Programming Notes | 484 |
| Sample Programs | 484 |

Appendix G. Notices 485

| | |
|----------------------|-----|
| Trademarks | 486 |
|----------------------|-----|

Index 489

Figures

| | | | |
|--|-----|---|-----|
| 1. Keystroke Flow | 25 | 6. IBM Workstation Requester and IBM Host Computer Server Relationship | 325 |
| 2. Host Presentation Space Characters | 50 | 7. Example of an SRPI Requester and Server Flow | 326 |
| 3. DDE Menu Server Conversation | 273 | 8. DDE Menu Server Conversation | 409 |
| 4. DDE Menu Client Conversation | 274 | 9. DDE Menu Client Conversation | 410 |
| 5. Example of PC/3270 SRPI Requester and Server | 324 | | |

Tables

| | | | |
|---|-----|--|-----|
| 1. Sample Program Files | 3 | 30. Base DDM Query Reply Format with Name and Direct Access Self-Defining Parameters | 333 |
| 2. Sample Program Subdirectories | 4 | 31. Base DDM Query Reply Format with Direct Access and Name Self-Defining Parameters | 333 |
| 3. EHLLAPI Return Codes. | 8 | 32. IBM Auxiliary Device Base Format with Direct Access Self-Defining Parameter | 335 |
| 4. EHLLAPI Read and Write Sharing Option Combinations | 15 | 33. IBM Auxiliary Device Direct Access Self-Defining Parameter | 336 |
| 5. Prerequisite Functions and Associated Dependent Functions | 15 | 34. IBM Auxiliary Device PCLK Self-Defining Parameter. | 336 |
| 6. EHLLAPI Functions Summary | 28 | 35. OEM Auxiliary Device Base Format with Direct Access Self-Defining Parameter | 336 |
| 7. Mnemonics with Uppercase Alphabetic Characters | 137 | 36. OEM Auxiliary Device Direct Access Self-Defining Parameter | 337 |
| 8. Mnemonics with Numbers or Lowercase Characters | 138 | 37. IBM Auxiliary Device PCLK Self-Defining Parameter. | 337 |
| 9. Mnemonics with @A and @ Uppercase Alphabetic Characters. | 138 | 38. CPR Query Reply Buffer Format | 338 |
| 10. Mnemonics with @A and @ Lowercase Alphabetic Characters. | 139 | 39. IBM Product-Defined Query Reply Base Format. | 339 |
| 11. Mnemonics with @A and @ Alphanumeric (Special) Characters | 140 | 40. Valid REFID and SSID Values for the IBM Product-Defined Query Reply | 339 |
| 12. Mnemonics with @S (Shift), @W (Edit) and @ Alphabetic Characters. | 140 | 41. IBM Product-Defined Direct Access Self-Defining Parameter | 340 |
| 13. Mnemonics Using @X and @Alphabetic Lowercase (For DBCS Only) | 140 | 42. IBM DIA Base Format. | 340 |
| 14. Mnemonics Using @M, @Q and @Alphabetic Lowercase (For VT Only) | 141 | 43. IBM Product-Defined Direct Access Self-Defining Parameter | 341 |
| 15. Mnemonics with Special Character Keys | 144 | 44. Type 1 Return Code Definitions and Descriptions | 350 |
| 16. BIDI Key Mnemonics | 144 | 45. Type 3 Return Code Definitions and Descriptions | 353 |
| 17. Naming Scheme for Data Items | 215 | 46. Class Definitions for Type 2 and Type 3 | 353 |
| 18. DDE Functions Available for Personal Communications | 217 | 47. Exception Code Values for Type 2 and Type 3 | 354 |
| 19. SENDKEY Command List | 247 | 48. Exception Object Values for Type 2 and Type 3. | 354 |
| 20. DDE Function Summary | 285 | 49. Naming Scheme for Data Items | 357 |
| 21. Naming Scheme for Data Items | 291 | 50. DDE Functions in a 16-Bit Environment | 358 |
| 22. Topics for Personal Communications | 291 | 51. SENDKEY Command List | 385 |
| 23. Mark Word Left. | 303 | 52. Summary of DDE Functions in a 16-Bit Environment. | 420 |
| 24. Mark Word Right | 303 | 53. Prerequisite Calls for Functions | 426 |
| 25. Parameters Supplied by the SRPI Requester | 326 | 54. EHLLAPI and REXX EHLLAPI Functions | 427 |
| 26. Parameters Returned to the SRPI Requester | 328 | | |
| 27. DDM Query Reply Base Format | 331 | | |
| 28. DDM Application Name Self-Defining Parameter. | 332 | | |
| 29. DDM PCLK Auxiliary Device Self-Defining Parameter. | 332 | | |

About This Book

This book provides necessary programming information for you to use the IBM® Personal Communications for Windows® Emulator High-Level Language Application Program Interface (EHLLAPI), Dynamic Data Exchange (DDE), Personal Communications Session API (PCSAPI), and Server-Requester Programming Interface (SRPI). The Host Access Class Library is described in *Host Access Class Library*.

EHLLAPI/DDE/PCSAPI is used with Personal Communications 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.

In this book, *Windows* refers to Windows 2003, Windows XP, Windows Vista, Windows 7 and Windows Server 2008. When information is relevant only to a specific operating system, this will be indicated in the text.

Who Should Read This Book

This book is intended for programmers who write application programs that use the APIs documented in this book.

A working knowledge of Windows is assumed. For information about Windows, refer to the list of publications under “Where To Find More Information.”

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 Where To Find More Information for the appropriate references for the specific language you are using.

Where To Find More Information

The Personal Communications library includes the following publications:

- *Installation Guide*
- *Quick Beginnings*
- *Emulator User's Reference*
- *Administrator's Guide and Reference*
- *Emulator Programming*
- *Client/Server Communications Programming*
- *System Management Programming*
- *CM Mouse Support User's Guide and Reference*
- *Host Access Class Library*
- *Configuration File Reference*

In addition to the printed books, there are Hypertext Markup Language (HTML) documents provided with Personal Communications:

Host Access Class Library

This HTML document describes how to write an ActiveX/OLE 2.0-compliant application to use Personal Communications as an embedded object.

Following is a list of related publications:

- *IBM 3270 Information Display System Data Stream Programmer's Reference*, GA23-0059
- *IBM 5250 Information Display System Functions Reference Manual*, SA21-9247

Refer to the IBM Glossary of Computing Terms at <http://www.networking.ibm.com/nsg/nsgmain.htm> for definitions of technical terms used throughout this book.

Support Options

If you determine that you need to contact IBM, you can do any of the following:

- Access the Personal Communications Web page at the following URL:

<http://www.ibm.com/software/network/pcomm>

- To find the phone number for IBM Software Support, U.S. customers can call 1-800-IBM-4YOU. International customers that have access to the U.S. "800" toll free numbers can reach the International Support Center by calling 1-800-IBM-4YOU and asking to speak with the International Support Center (ISC) in Atlanta. International customers without access to the U.S. toll free numbers can call the ISC directly at 770-863-1234. The ISC's FAX number is 770-863-3030 and is available 24 hours a day.

Subscribing to Support News

To receive Personal Communications support news flashes, complete the following steps:

1. Go to the Personal Communications support Web site, at the following address: <http://www.ibm.com/software/network/pcomm/support>
2. Under **Personal Support**, click **My Support**. If you are already have an IBM Registration ID, you can sign in. If not, you must register.
3. After you sign in, select **Networking** from the product family list.
4. Select the products for which you would like to receive news flashes. Click **Save and Return**. Your support profile is listed on the next screen.
5. For your mail preference, you can choose to receive e-mail flashes, downloadable files, or both. Click **Submit**. You will receive a message indicating that your profile has been updated.
6. Click **Sign out** to end the session.

Support Assistant

The IBM Support Assistant enables you to easily resolve software questions. The Support Assistant provides the following components:

- **Search**
Enable search of the software information database.
- **Service**
Assists customers who choose to submit a PMR by providing access to the Electronic Service Request Web site.

- **Support Links**

A consolidated list of IBM web links, organized by brand and product.

Help desk personnel and Personal Communications administrators might want to install Support Assistant in order to better support end users. The Support Assistant can be downloaded from the following address: <http://www.ibm.com/software/support/isa/index.html>

Notation

A table at the beginning of each section explains API or DDE functions in Chapter 3, “EHLAPI Functions,” on page 27, Chapter 5, “PCSAPI Functions,” on page 191, Chapter 6, “DDE Functions in a 32-bit Environment,” on page 215, and Appendix E, “DDE Functions in a 16-Bit Environment,” on page 357. It shows whether a function is supported for the products that provide the function described in the section. Yes means it is supported for a host type, and No means not supported. For example, the following table indicates that a function is available for 3270 and VT sessions but not for 5250 sessions.

| <i>3270</i> | <i>5250</i> | <i>VT</i> |
|-------------|-------------|-----------|
| Yes | No | Yes |

Chapter 1. Introduction to Emulator APIs

The IBM 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 27.
 - **IBM Standard HLLAPI Support:** This is a standard programming interface which allows programmatic access to a host emulator session. See Chapter 2, "Introduction to IBM Standard EHLLAPI, IBM Enhanced EHLLAPI and WinHLLAPI Programming," on page 5.
 - **IBM Enhanced HLLAPI Support:** This interface is based on the IBM Standard HLLAPI interface. It provides all of the existing functionality but uses modified data structures. See Chapter 2, "Introduction to IBM Standard EHLLAPI, IBM Enhanced EHLLAPI and WinHLLAPI Programming," on page 5.
 - **Windows High Level Language API (WinHLLAPI):** This interface provides much of the same functionality of IBM Standard EHLLAPI and adds some extensions that take advantage of the Windows environment. See Chapter 2, "Introduction to IBM Standard EHLLAPI, IBM Enhanced EHLLAPI and WinHLLAPI Programming," on page 5.
 - **REXX EHLLAPI:** This allows programmers who are using EHLLAPI to write REXX language application programs.
- **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 6, "DDE Functions in a 32-bit Environment," on page 215.
- **Personal Communications Session API (PCSAPI):** This interface is used to start, stop, and control emulator sessions and settings. See Chapter 5, "PCSAPI Functions," on page 191.

For Personal Communications Version 6.0, functions have been added to allow control and retrieval of page and printer settings. See "Page Setup Functions" on page 199 and "Printer Setup Functions" on page 205.

- **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 8, "Server-Requestor Programming Interface (SRPI) Support," on page 323.

- **IBM 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)). Refer to *Host Access Class Library (HACL)* for more details.

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 (**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 you exit from the critical section.

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.

Stack Size

Emulator APIs use the calling program's 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 ensure sufficient stack space is available for the API.

Running 16-bit Windows EHLLAPI programs

If you are running multiple 16-bit Windows EHLLAPI tasks that use any of the 16-bit EHLLAPI DLLs, each 16-bit EHLLAPI task must run under a separate NTVDM. You can use any of the following methods to accomplish this:

- Shortcuts used to start 16-bit EHLLAPI applications must specify that the program run in a separate memory space (NTVDM).
- To start a 16-bit EHLLAPI application such as `hllapi16.exe` from a command prompt or batch program, type the following command:
`start /separate hllapi16.exe`
- If a Win32 application spawns a 16-bit EHLLAPI application using the Windows API `CreateProcess`, it must use the Process creation flag named `CREATE_SEPARATE_WOW_VDM`.

Windows x64 Platform Support

The x64-based versions of Microsoft® Windows Server 2003 and Microsoft Windows XP Professional x64 Edition are optimized to run native 64-bit programs, but do not support 32-bit drivers or 16-bit applications.

For these platforms, Personal Communications does not install the following libraries.

- DOS EHLLAPI
- 16-bit API support:
 - Standard EHLLAPI 16-bit interface
 - WinHLLAPI 16-bit interface
 - PCSAPI 16-bit interface
 - SRPI 16-bit interface

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 \SAMPLES directory.

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 program files include source and supporting files for the following Personal Communications APIs:

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

The following files are installed in the \SAMPLES directory.

Table 1. Sample Program Files

| File Name | Description |
|--------------|--|
| DDE_C.H | DDE include file |
| EHLAPI32.H | IBM standard 32-bit EHLLAPI include file |
| WHLLAPI.H | WinHLLAPI 16-bit include file |
| HAPI_C.H | EHLLAPI include file |
| PCSAPI.H | PCSAPI include file |
| PCSCALLS.LIB | Import library for standard interface |
| PCSCAL32.LIB | Import library for enhanced interface |
| EHLAPI32.LIB | Import library for IBM Standard 32-bit EHLLAPI interface |
| WHLLAPI.LIB | Import library for WinHLLAPI 16-bit interface |
| WHLAPI32.LIB | Import library for WinHLLAPI 32-bit interface |
| UCCPRB.H | SRPI include file |

The following subdirectories are created in the \SAMPLES directory.

Table 2. Sample Program Subdirectories

| File Name | Description |
|-----------|---|
| DDXFER | Shows how EHLLAPI can be used to create a “Drag and Drop” application; in this case, for file transfer |
| ECL | HACL sample files |
| HLLSMP | Shows how to use EHLLAPI to request a keystroke and log on to a VM system |
| LISTFILE | Illustrates how DDE can make use of the LOAD button to transfer files from the host |
| PCSMAN | Illustrates the use of PCSAPI to start and stop sessions, query the session status, and query the profile for the session |
| SPL2FILE | A program that uses DDE to save an iSeries®, eServer™ i5, or System i5® spool file as an ASCII file on the PC |
| SRPSMP | Illustrates the use of the Server Requester Programming Interface (SRPI) |
| VBDDE | VBDDE sample files |
| VBHLLAPI | VBHLLAPI sample files |
| VBPCSAPI | VBPCSAPI sample files |

Displaying Arabic data in the VBHLLAPI sample program

For proper display of Arabic data in the VBHLLAPI sample program, you should do the following:

1. Edit the VBHLLAPI sample program source code. For each form, change the default font property to a Windows True Type font that supports Arabic script (for example, Courier New).
2. Recompile the VBHLLAPI sample program.

Chapter 2. Introduction to IBM Standard EHLLAPI, IBM Enhanced EHLLAPI and WinHLLAPI Programming

This chapter provides information needed to incorporate IBM Standard EHLLAPI (16- and 32-bit), WinHLLAPI (16- and 32-bit), and IBM Enhanced 32-bit EHLLAPI (EHLAPI32) functions into applications written in a high level language. It provides details on call format, memory allocation considerations, initializing the interfaces, and compiling and linking 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 Overviews

Following are overviews for HLLAPI programming interfaces.

IBM Standard EHLLAPI

EHLLAPI is a standard 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).

WinHLLAPI

WinHLLAPI is based on the familiar EHLLAPI.API. It encompasses all of the existing functionality and adds extensions that take advantage of the Windows message driven environment. Users of the IBM Personal Communications EHLLAPI interface will notice no functional difference unless they incorporate the WinHLLAPI extensions.

The WinHLLAPI extension functions and any functions that deviate from the EHLLAPI form are described in Chapter 4, "WinHLLAPI Extension Functions," on page 179. For information on common functions, refer to Chapter 3, "EHLLAPI Functions," on page 27.

WinHLLAPI and IBM Standard EHLLAPI

The entry symbol for WinHLLAPI, is appropriately, **WinHLLAPI**. EHLLAPI users wishing to switch to the WinHLLAPI implementation must change from the **hllapi** standard entry. New users should follow all of the directions in Chapter 3, "EHLLAPI Functions," on page 27, and use the **WinHLLAPI** entry in place of the standard **hllapi** entry.

IBM Enhanced EHLLAPI and IBM Standard EHLLAPI

IBM Enhanced EHLLAPI is based on the familiar EHLLAPI API. It encompasses all of the existing functionality but takes advantage of the 32-bit environment and uses modified data structures. Standard interface users wishing to switch to IBM Enhanced 32-bit EHLLAPI need to change only the entry symbol from LPWORD to LPINT in the first, third, and fourth parameters. New users should use the procedures in the following sections.

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++ 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 VisualAge[®] for C/C++
- Microsoft Visual C/C++ Version 4.0 and higher

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 "Standard and Enhanced Interface Considerations" on page 21).

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. Position (input); Return Code (output)

The prototype for IBM Standard EHLLAPI is:

```
[long hllapi (LPWORD, LPSTR, LPWORD, LPWORD);
```

The prototype for IBM Enhanced EHLLAPI is:

```
[long hllapi (LPINT, LPSTR, LPINT, LPINT);
```

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;
long   Rc;

memset(QueryData, 0, sizeof(QueryData)); // Init buffer
QueryData.qsst_shortcode = '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 a **long** data type for IBM Standard and Enhanced EHLLAPI, and **void** data type for WinHLLAPI, 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 the same for all IBM Standard and WinHLLAPI 16- and 32-bit applications. Data structures for the IBM Enhanced 32-bit applications are packed to a 4-byte alignment.

It is *highly recommended* that the supplied header file and data structure definitions be used to ensure 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 session
}

```

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 session
}

```

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 pre-allocated 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 or return 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.

Table 3. EHLLAPI Return Codes

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

Table 3. EHLLAPI Return Codes (continued)

| Return Code | Explanation |
|-------------|--|
| 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). |

Compiling and Linking

Applications using EHLLAPI functions must include the appropriate header file to obtain the proper function prototypes, constants, and data structure definitions. These header files may be used with any of the supported C/C++ compilers (see “Languages” on page 6). 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 entry point is to be resolved. The simplest way is to statically link the application with the appropriate Personal Communications library. This will resolve the entry point at link time. The operating system will load the correct DLL with the application when it starts. Another way to link to the entry point is to perform dynamic linking. In this case, the application uses operating system calls to load the correct DLL and obtain the entry point address at run time.

The following table shows which header files to use, which .LIB should be used for static linking, and which .DLL should be used for dynamic loading.

| Interface | Entry Point | Header File | LIB | DLL |
|-----------------------|-------------|-------------|--------------|--------------|
| IBM Standard (16-bit) | hllapi | hapi_c.h | PCSCALLS.DLL | PCSHLL.DLL |
| IBM Standard (32-bit) | hllapi | ehllapi32.h | EHLAPI32.LIB | EHLAPI32.DLL |
| IBM Enhanced (32-bit) | hllapi | hapi_c.h | PCSCAL32.LIB | PCSHLL32.DLL |
| WinHLLAPI (16-bit) | winhllapi | whllapi.h | WHLLAPI.LIB | WHLLAPI.DLL |
| WinHLLAPI (32-bit) | winhllapi | whllapi.h | WHLAPI32.LIB | WHLAPI32.DLL |

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(&HFunc, HBuff, &HLen, &HRc);
if (HRc != 0) {
    // ... EHLLAPI access error
}
```

When the application is linked, the appropriate Personal Communications library files must be linked with the application executable code. For example, the following link command might be used (IBM VisualAge C/C++):

```
ilink /de /noe pcsca132.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 appropriate Personal Communications module and locate the 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 header file. The following 32-bit Windows code loads the IBM Standard 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
}

```

Multithreading

IBM Enhanced EHLLAPI (32-bit) and IBM Standard EHLLAPI 16-bit connect on a per process basis. All threads access the same connected host session. The thread that performs the connections must also perform the disconnection.

IBM Standard EHLLAPI (32-bit) and WinHLLAPI connect on a per thread basis. Each thread must maintain its own connections. This allows a multithreaded process to maintain connections to more than one connected host session at a time. This eliminates the need for multi-process schemes when using a WinHLLAPI program to coordinate data between different hosts. It also puts the burden of connecting and disconnecting as necessary on the individual thread.

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.

IBM Enhanced 32-Bit Interface Presentation Space IDs

For IBM Enhanced EHLLAPI applications, the session ID is extended with three 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_shortcode = '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 IDs

EHELLAPI functions interact with only one presentation space at a time. The presentation space ID (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.
- There is no default 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)**
- **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:

- The function can interact with any host presentation space if a letter, not a blank or a null, is used to specify the PSID.
- If a blank or a null is used to specify the PSID, a return code of 1 is generated. The function does not execute.
- Using a letter to specify the PSID does not establish a host-connected presentation space, except on a connect PS request.

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

- The function can interact with any host presentation space if a letter is used to specify the PSID.
- 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.

- Using a letter to specify the PSID does not change the established PSID of the host-connected presentation space, except on a connect PS request.

The following functions are available for printer sessions:

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

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 behavior¹). 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

1. This means that two EHLLAPI programs will not be vying for the same Presentation Space at the same time; or that there is logic in those programs which will allow the program to wait until the PS is available; or that the applications never use the Session in a way which would lock out other applications.

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 4. 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 non-shareable. 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 5. Prerequisite Functions and Associated Dependent Functions

| Prerequisite Call | Functions | Access |
|--------------------------------------|--|---|
| Allocate Communications Buffer (120) | Free Communication Buffer (120) | N/A |
| 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 |

Table 5. Prerequisite Functions and Associated Dependent Functions (continued)

| Prerequisite Call | Functions | Access |
|--------------------------------|---|---|
| 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 Read Write Write Read |
| Connect Structured Field (120) | Disconnect Structured Field (121) Get Request Completion (125) Read Structured Field (126) Write Structured Field (127) | N/A |
| Read Structured Field (126) | Get Request Completion (125) | N/A |
| Start Close Intercept (41) | Query Close Intercept (42) Stop Close Intercept (43) | N/A |
| 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) Send Key (3) if edit keystrokes are to be sent (edit keystroked support is available in Enhanced Mode) | N/A |
| Write Structured Field (127) | Get Request Completion (125) | N/A |

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.

Using mouse actions to select, copy, and paste text in the Presentation Space

The following mouse actions can be used in the Presentation Space.

- Select a word by double-clicking the left mouse button.

- Copy a selected word by clicking the right mouse button.
- Paste a copied word by double-clicking the mouse right button.

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 lists:

| | | | | | |
|----|--------------|----|----------|-----|-------------------|
| @B | Left Tab | @0 | Home | @h | PF17 |
| @C | Clear | @1 | PF1/F1 | @i | PF18 |
| @D | Delete | @2 | PF2/F2 | @j | PF19 |
| @E | Enter | @3 | PF3/F3 | @k | PF20 |
| @F | Erase EOF | @4 | PF4/F4 | @l | PF21 |
| @H | Help (PC400) | @5 | PF5/F5 | @m | PF22 |
| @I | Insert | @6 | PF6/F6 | @n | PF23 |
| @J | Jump | @7 | PF7/F7 | @o | PF24 |
| @L | Cursor Left | @8 | PF8/F8 | @q | End |
| @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 |
| @Z | Cursor Right | | | | |

| | | | |
|------|-------------------------------|------|--|
| @A@C | Test (PC400) | @A@e | Pink (PC/3270) |
| @A@D | Word Delete | @A@f | Green (PC/3270) |
| @A@E | Field Exit | @A@g | Yellow (PC/3270) |
| @A@F | Erase Input | @A@h | Blue (PC/3270) |
| @A@H | System Request | @A@i | Turquoise (PC/3270) |
| @A@I | Insert Toggle | @A@j | White (PC/3270) |
| @A@J | Cursor Select | @A@l | Reset Host Color (PC/3270) |
| @A@L | Cursor Left Fast | @A@t | Print (Personal Computer) |
| @A@Q | Attention | @A@u | Rollup (PC400) |
| @A@R | Device Cancel | @A@v | Rolldown (PC400) |
| @A@T | Print Presentation Space | @A@y | Forward Word Tab |
| @A@U | Cursor Up Fast | @A@z | Backward Word Tab |
| @A@V | Cursor Down Fast | @A@- | Field - (PC400) |
| @A@Z | Cursor Right Fast | @A@+ | Field + (PC400) |
| @A@9 | Reverse Video | @A@< | Record Backspace (PC400) |
| @A@b | Underscore (PC/3270) | @S@E | Print Presentation Space on Host (PC400) |
| @A@c | Reset Reverse Video (PC/3270) | @S@x | Dup |
| @A@d | Red (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 prefixes all characters and mnemonics sent to the host application with two ASCII characters. The first ASCII character is the PSID of the host presentation space to which the keystrokes are sent. The other character is an A, S, or M for ASCII, special shift, or mnemonic, respectively. See "Return Parameters" on page 89.

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, refer to *Administrator's Guide and Reference*.

A Simple EHLLAPI Sample Program

The following sample Windows 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 HLDCConnectPS ConnBuff;
    // Send Key string for HOME+string+ENTER:
```

```

char SendString[] = "@@Hello World!@E";

HFunc = HA_RESET_SYSTEM;
HLen = 0;
HRc = 0;
hllapi(&HFunc, HBuf, &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_shortcode = '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);
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, HBuf, &HLen, &HRc);

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

```

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

```

all: sample.exe

hllldir = C:\PCOMWIN\SAMPLES
hlllib = C:\PCOMWIN\SAMPLES

.SUFFIXES: .C .OBJ

.c.obj:
    icc.exe /Ti /Gh /Gm /Gd /C /I $(hllldir) /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
```

Standard and Enhanced Interface Considerations

There is no functional difference between the standard and enhanced EHLLAPI interfaces on a given platform. However there are other important differences:

- The enhanced 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 ensure compatibility with future versions of enhanced EHLLAPI.
- The position (offset) of data elements in memory buffers passed to and from EHLLAPI functions are different. Data elements in enhanced EHLLAPI are aligned to double-word boundaries. Data elements in standard 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 ensure 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 standard or enhanced operation without any source code changes. For example, the following section of code would work for standard EHLLAPI but would fail for enhanced 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 enhanced 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 standard or enhanced operation. Changed lines are indicated with a >:

```

#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_shortcode = 'A';     // Session to query
hllapi(&Func, (char *)&Buff, &Len, &Rc); // Execute function

> SessType = Buff.qsst_sesstype; // 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 time-out 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 "General" on page 17 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 Personal Communications 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 provides a simplified representation of the keystroke flow and the objects within a keyboard enhancement environment.

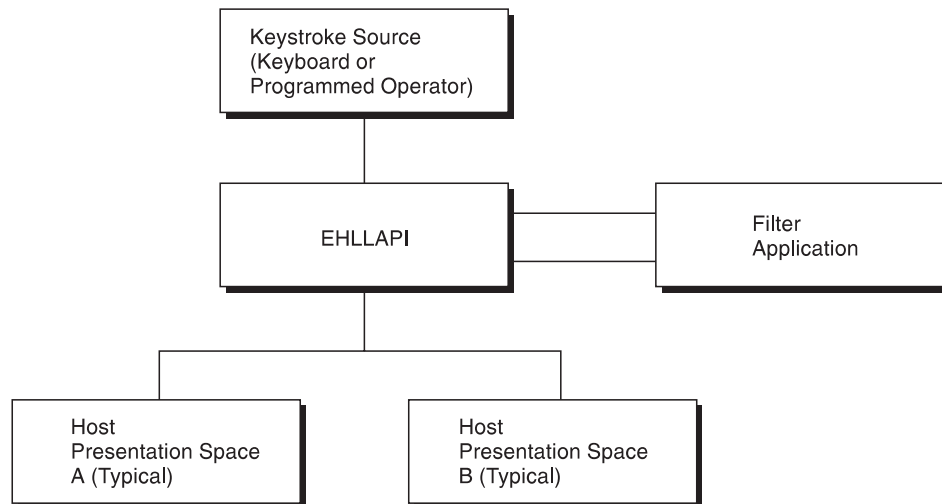


Figure 1. Keystroke Flow

Scenario 7. Keyboard Enhancement

This scenario 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 Personal Communications 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 standard and enhanced interfaces.

Note: Throughout this chapter WinHLLAPI, IBM Standard 32-bit HLLAPI and 16-bit EHLLAPI are referred to as Standard Interface, and IBM Enhanced 32-bit EHLLAPI is referred to as Enhanced Interface.

Unicode Support for Code Pages 1390/1399 and 1137

The following EHLLAPI functions are enabled for Japanese code page 1390/1399 and Hindi code page 1137 support on a Unicode session:

- Convert Position or Convert RowCol (1137 only)
- Copy Field to String
- Copy Presentation Space
- Copy Presentation Space to String
- Copy String to Field
- Copy String to Presentation Space
- Get Key
- Search Field
- Search Presentation Space
- Send Key
- Set Cursor (1137 only)
- Set Session Parameters

See the specific section for each function for details on Japanese code page 1390/1399 and Hindi code page 1137.

Notes:

1. The string containing the Unicode characters to be sent to the PCOMM session should be typecast to WCHAR * for code page 1390/1399 and to char * for code page 1137.
2. EHLLAPI 1390/1399 Unicode functionality is available only for 3270 and 5250 sessions. EHLLAPI 1137 Unicode functionality is available only for 5250 sessions.

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.

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

Table 6 is the summary of the EHLLAPI functions:

Table 6. EHLLAPI Functions Summary

| Function | 3270 | 5250 | VT |
|---|------|------|-----|
| “Connect Presentation Space (1)” on page 36 | Yes | Yes | Yes |
| “Disconnect Presentation Space (2)” on page 82 | Yes | Yes | Yes |
| “Send Key (3)” on page 135 | Yes | Yes | Yes |
| “Wait (4)” on page 169 | Yes | Yes | Yes |
| “Copy Presentation Space (5)” on page 57 | Yes | Yes | Yes |
| “Search Presentation Space (6)” on page 129 | Yes | Yes | Yes |
| “Query Cursor Location (7)” on page 107 | Yes | Yes | Yes |
| “Copy Presentation Space to String (8)” on page 64 | Yes | Yes | Yes |
| “Set Session Parameters (9)” on page 147 | Yes | Yes | Yes |
| “Query Sessions (10)” on page 112 | Yes | Yes | Yes |
| “Reserve (11)” on page 123 | Yes | Yes | Yes |
| “Release (12)” on page 123 | Yes | Yes | Yes |
| “Copy OIA (13)” on page 48 | Yes | Yes | Yes |
| “Query Field Attribute (14)” on page 107 | Yes | Yes | Yes |
| “Copy String to Presentation Space (15)” on page 76 | Yes | Yes | Yes |
| “Pause (18)” on page 100 | Yes | Yes | Yes |
| “Query System (20)” on page 113 | Yes | Yes | Yes |
| “Reset System (21)” on page 124 | Yes | Yes | Yes |
| “Query Session Status (22)” on page 110 | Yes | Yes | Yes |
| “Start Host Notification (23)” on page 160 | Yes | Yes | Yes |
| “Query Host Update (24)” on page 109 | Yes | Yes | Yes |
| “Stop Host Notification (25)” on page 168 | Yes | Yes | Yes |

Table 6. EHLLAPI Functions Summary (continued)

| Function | 3270 | 5250 | VT |
|--|------|------|-----|
| "Search Field (30)" on page 125 | Yes | Yes | Yes |
| "Find Field Position (31)" on page 86 | Yes | Yes | Yes |
| "Find Field Length (32)" on page 85 | Yes | Yes | Yes |
| "Copy String to Field (33)" on page 72 | Yes | Yes | Yes |
| "Copy Field to String (34)" on page 40 | Yes | Yes | Yes |
| "Set Cursor (40)" on page 146 | Yes | Yes | Yes |
| "Start Close Intercept (41)" on page 156 | Yes | Yes | Yes |
| "Query Close Intercept (42)" on page 104 | Yes | Yes | Yes |
| "Stop Close Intercept (43)" on page 166 | Yes | Yes | Yes |
| "Query Additional Field Attribute (45)" on page 103 | No | Yes | No |
| "Start Keystroke Intercept (50)" on page 163 | Yes | Yes | Yes |
| "Get Key (51)" on page 89 | Yes | Yes | Yes |
| "Post Intercept Status (52)" on page 102 | Yes | Yes | Yes |
| "Stop Keystroke Intercept (53)" on page 168 | Yes | Yes | Yes |
| "Lock Presentation Space API (60)" on page 97 | Yes | No | No |
| "Lock Window Services API (61)" on page 99 | Yes | No | No |
| "Start Communication Notification (80)" on page 158 | Yes | Yes | Yes |
| "Query Communication Event (81)" on page 106 | Yes | Yes | Yes |
| "Stop Communication Notification (82)" on page 167 | Yes | Yes | Yes |
| "Send File (90)" on page 183 | Yes | Yes | No |
| "Receive File (91)" on page 121 | Yes | Yes | No |
| "Cancel File Transfer (92)" on page 31 | Yes | Yes | Yes |
| "Convert Position or Convert RowCol (99)" on page 38 | Yes | Yes | Yes |
| "Connect Window Services (101)" on page 37 | Yes | Yes | Yes |
| "Disconnect Window Service (102)" on page 82 | Yes | Yes | Yes |
| "Query Window Coordinates (103)" on page 115 | Yes | Yes | Yes |
| "Window Status (104)" on page 170 | Yes | Yes | Yes |
| "Change Switch List LT Name (105)" on page 33 | Yes | Yes | Yes |
| "Change PS Window Name (106)" on page 32 | Yes | Yes | Yes |
| "Start Playing Macro (110)" on page 165 | Yes | Yes | Yes |
| "Connect for Structured Fields (120)" on page 34 | Yes | No | No |
| "Disconnect from Structured Fields (121)" on page 80 | Yes | No | No |
| "Query Communications Buffer Size (122)" on page 104 | Yes | No | No |
| "Allocate Communications Buffer (123)" on page 30 | Yes | No | No |
| "Free Communications Buffer (124)" on page 88 | Yes | No | No |
| "Get Request Completion (125)" on page 94 | Yes | No | No |
| "Read Structured Fields (126)" on page 116 | Yes | No | No |
| "Write Structured Fields (127)" on page 173 | Yes | No | No |

Allocate Communications Buffer (123)

| | | |
|------|------|----|
| 3270 | 5250 | VT |
| Yes | No | No |

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

Prerequisite Calls

There are no prerequisite calls for this function.

Call Parameters

| | Standard Interface | Enhanced Interface |
|-----------------|-------------------------|--------------------|
| 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:

| Byte | | Definition |
|----------|----------|--|
| Standard | Enhanced | |
| 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

- The EHLLAPI obtains a buffer from the operating system 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.
- 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.

Cancel File Transfer (92)

| | | |
|------|------|-----|
| 3270 | 5250 | VT |
| Yes | Yes | Yes |

The **Cancel File Transfer** function causes any current EHLLAPI initiated **Send File** or **Receive File** for the specified session to immediately return.

Prerequisite Calls

Send File (90) or **Receive File** (91)

Call Parameters

| | Enhanced Interface |
|-----------------|---|
| Function Number | Must be 92 |
| Data String | 1-character short name of the host presentation space. A blank or null indicates request for updates to the host-connected presentation space |
| Length | 4 is implied |
| PS Position | NA |

The calling data structure contains these elements

| Byte | Definition |
|------|--|
| 1 | A 1-character presentation space short name (PSID) |
| 2-4 | Reserved |

Return Parameters

| Return Code | Definition |
|-------------|--|
| 0 | The function was successful |
| 1 | An incorrect PSID was specified |
| 8 | No prior call to Start Communication Notification (80) function was called for the PSID |
| 9 | A system error was encountered |

Notes on Using This Function

Since both **Send File** (90) and **Receive File** (91) are blocking calls, this function must always be issued on a different thread.

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

| | Standard Interface | Enhanced Interface |
|-----------------|-------------------------------|--------------------|
| Function Number | Must be 106 | |
| Data String | See the following table | |
| 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 enhanced interface).

The calling data string can contain:

| Byte | | Definition |
|----------|----------|---|
| Standard | Enhanced | |
| 1 | 1 | A 1-character presentation space short name (PSID) |
| | 2–4 | Reserved |
| 2 | 5 | A change request option value, select one of: <ul style="list-style-type: none"> • 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. |

| Return Code | Explanation |
|-------------|----------------------|
| 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 Communication Manager EHLLAPI, and has the same result as the **Change PS Window Name** (106) function.

Prerequisite Calls

Connect Window Services (101)

Call Parameters

| | Standard Interface | Enhanced Interface |
|-----------------|-------------------------|--------------------|
| 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:

| Byte | | Definition |
|----------|----------|--|
| Standard | Enhanced | |
| 1 | 1 | A 1-character presentation space short name (PSID) |
| | 2–4 | Reserved |
| 2 | 5 | A change request option; select: <ul style="list-style-type: none"> X'01' for changing a switch list LT name X'02' for resetting a switch list LT name |

| Byte | | Definition |
|------|-------|---|
| 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. |
| 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

There are no prerequisite calls for this function.

Call Parameters

| | Standard Interface | Enhanced Interface |
|-----------------|-------------------------|--------------------|
| 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:

| Byte | | Definition |
|----------|----------|---|
| Standard | Enhanced | |
| 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

- 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.
- 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," on page 331, for the detailed formats and usages of the query reply data structures supported by EHLLAPI.
- Only cursory checking is performed on the Query Reply Data. Only the ID and the length of the structure are checked for validity.
- 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.
- If return code RC=32 or RC=39 is received, an application is already connected to the selected session and use of that presentation space should be approached with caution. Conflicts with SRPI, file transfer, and other EHLLAPI applications might result.

Connect Presentation Space (1)

| | | |
|------|------|-----|
| 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

There are no prerequisite calls for this function.

Call Parameters

| | Standard Interface | Enhanced Interface |
|-----------------|---|--------------------|
| 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:

| Byte | | Definition |
|----------|----------|--|
| Standard | Enhanced | |
| 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. 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 147.
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. You cannot connect to a session that is defined as a logical printer session. Refer to *Administrator’s Guide and Reference* for more information.

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

There are no prerequisite calls for this function.

Call Parameters

| | Standard Interface | Enhanced Interface |
|-----------------|---|--------------------|
| 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:

| Byte | | Definition |
|----------|----------|--|
| Standard | Enhanced | |
| 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. |
| 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.
2. 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.

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

There are no prerequisite calls for this function.

Call Parameters

| | Standard Interface | Enhanced Interface |
|-----------------|--------------------|--------------------|
| Function Number | Must be 99 | |

| | Standard Interface | Enhanced Interface |
|-------------|---|--------------------|
| Data String | Host presentation space short name <i>and</i> P for the Convert Position function (for example, AP converts the presentation space position of session A); <i>or</i> 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 | <p>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 40.</p> <p>NA when P is specified as the second character in the data string parameter.</p> | |
| PS Position | <p>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 40.</p> <p>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 40.</p> | |

The calling data string can contain:

| Byte | | Definition |
|----------|----------|--|
| Standard | Enhanced | |
| 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 *Administrator's Guide and Reference*
2. To find out how many rows and columns are in your presentation space, examine the returned data string parameter for the **Query Session Status (22)** function. See "Query Session Status (22)" on page 110.

1137 Code Page Support

Unicode functionality is supported only on 5250 sessions.

Convert Position or **Convert RowCol** is Hindi enabled in order to return the beginning of the cluster. The usage of **Convert Position** or **Convert RowCol** is the same as the SBCS session.

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

| | Standard Interface | Enhanced Interface |
|-----------------|---|--------------------|
| 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, length, 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

Length:

The length of the data returned.

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. |
| 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 EAB/NOEAB, the XLATE/NOXLATE, the DISPLAY/NODISPLAY, the DISPLAY/NODISPLAY, the EAD/NOEAD (for DBCS only), and the NOS0/SPACES0/S0 (for DBCS only) session options. Refer to items 5 on page 149; 13 and 14 on page 152; 17 on page 153; and 20 and 21 on page 154 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 left most 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 Personal Communications 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 |

| Bit Position | Character Position | Field Attribute Position |
|--------------|---------------------|---|
| 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) |
| 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.

- 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 8 for more information.

Note: 5250 emulation supports a presentation space of 24 rows by 80 columns. In some instances, Communication 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 displays 25th row information on the status bar. By **EXTEND_PS** option, an EHLLAPI application can use the same interface with Communication Manager EHLLAPI and valid presentation space is extended when this condition occurs.

1390/1399 Code Page Support

Unicode functionality is supported only on 3270 and 5250 sessions.

In a Unicode session, the characters in the host source presentation space are translated into Unicode. Attribute bytes are normally translated into blanks.

The XLATE option (that can be specified using the **Set Session Parameters** function) is not supported in a Unicode session. This means that even if this option is issued, the EABs will not be translated to the PC color graphics adapter (CGA) format.

Prerequisite Calls: Connect Presentation Space (1)

Call Parameters:

| | Standard Interface | Enhanced Interface |
|-----------------|---|--------------------|
| 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 EBCDIC field. | |
| Length | The length of the target data string in Unicode characters. | |
| 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, length, and a return code.

Data String:

String containing the Unicode data is returned.

Length:

Number of Unicode characters copied into string.

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. |
| 9 | A system error was encountered. |
| 24 | Unformatted host presentation space. |

Notes on Using This Function: The following options are supported in a Unicode session for **Copy Field To String** (34) and function in the same way as in DBCS:

- NOATTRB
- ATTRB
- NULLATTRB
- EAB
- NOEAB
- NOXLATE
- DISPLAY
- NODISPLAY

1137 Code Page Support

Unicode functionality is supported only on 5250 sessions.

In a Unicode session, the characters in the host source presentation space are translated into Unicode. Attribute bytes are normally translated into blanks.

The XLATE option (that can be specified using the **Set Session Parameters** function) is not supported in a Unicode session. This means that even if this option is issued, the EABs will not be translated to the PC color graphics adapter (CGA) format.

Prerequisite Calls: **Connect Presentation Space** (1)

Call Parameters:

| | Standard Interface | Enhanced Interface |
|-----------------|---|--------------------|
| Function Number | Must be 34 | |
| Data String | Preallocated target data string. The length should be twice the number of EBCDIC bytes required to be copied from the presentation space. 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 four times the length of the EBCDIC field. | |
| Length | The length of the target data string in bytes. This length should be at least 2 in a Unicode session. If not, an error code of 2 is returned. | |
| 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, length, and a return code.

Data String:

String containing the Unicode data is returned.

Length:

Number of Unicode characters copied into string. To get the number of bytes, multiply by 2.

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. |
| 9 | A system error was encountered. |
| 24 | Unformatted host presentation space. |

Notes on Using This Function: The following options are supported in a Unicode session for **Copy Field To String** and function in the same way as in SBCS:

- NOATTRB
- ATTRB
- NULLATTRB
- EAB
- NOEAB
- NOXLATE
- DISPLAY
- NODISPLAY

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

| | Standard Interface | Enhanced Interface |
|-----------------|---------------------------------|--------------------|
| 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” 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 8 for more information.

Format of the Returned OIA Data String

The OIA data string contains the following information:

| Byte | | Definition |
|----------|----------|--|
| Standard | Enhanced | |
| 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 *Quick Beginnings* 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, 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). 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.

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

Figure 2. Host Presentation Space Characters

- Group 1 (Offset 82): Online and Screen Ownership

| Bit | Meaning |
|-----|----------|
| 0–1 | Reserved |

| Bit | Meaning |
|-----|-----------------------------|
| 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 |
| 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 |
| 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 |
|-----|----------------------|
| 0-6 | 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 | Communications Error | |
| 1-5 | 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 EHLAPI 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

| | Standard Interface | Enhanced Interface |
|-----------------|---|--------------------|
| Function Number | Must be 5 | |
| Data String | <p>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.</p> <p>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 presentation space.</p> | |
| Length | NA (the length of the host presentation space is implied). | |
| PS Position | NA. | |

Return Parameters

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

Data String:

Contents of the connected host presentation space.

Length:

Length of the data copied.

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

- 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.
- 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.
- 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 (for DBCS only)
 - NOSO/SPACESO/SO (for DBCS only)
 - EXTEND_PS/NOEXTEND_PS

Refer to items 5 on page 149; 13, 14, 15 and 17 on page 153; and 20 and 21 on page 154 for more information.

If the target data string provided is not long enough to hold the requested data, unpredictable results can occur.

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 left most 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 |

| Bit Position | Meaning |
|--------------|---|
| 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 Personal Communications 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 <ul style="list-style-type: none"> • When the value of bit position 7 is 0: <ul style="list-style-type: none"> 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: <ul style="list-style-type: none"> 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 <ul style="list-style-type: none"> 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 8 for more information.

Note: 5250 emulation supports a presentation space of 24 rows by 80 columns. In some instances, Communication 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 displays 25th row information on row 24, or on the status bar. For information to be displayed on the status bar, the status bar must be configured. Refer to *Quick Beginnings* for information on configuring the status bar. By the **EXTEND_PS** option, an EHLLAPI application can use the same interface with Communication Manager EHLLAPI and valid presentation space is extended when this condition occurs.

1390/1399 Code Page Support

Unicode functionality is supported only on 3270 and 5250 sessions.

In a Unicode session, the characters in the host source presentation space are translated into Unicode. Attribute bytes are normally translated into blanks.

The XLATE option (that can be specified using the **Set Session Parameters** (9) function) is not supported in a Unicode session. This means that even if this option is issued, the EABs will not be translated to the PC color graphics adapter (CGA) format.

Prerequisite Calls: **Connect Presentation Space** (1)

Call Parameters:

| | Standard Interface | Enhanced Interface |
|-----------------|---|--------------------|
| Function Number | Must be 5 | |
| Data String | Preallocated target Unicode string. When the Set Session Parameters (9) function with Extended Attribute Bytes (EAB) option is issued, the length of the data string must be twice the size of the presentation 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:

String containing the Unicode representation of the contents of presentation space is returned

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: The following options are supported in a Unicode session for **Copy Presentation Space (5)** and function in the same way as in DBCS:

- NOATTRB
- ATTRB
- NULLATTRB
- EAB
- NOEAB
- NOXLATE
- DISPLAY
- NODISPLAY
- BLANK
- NOBLANK

1137 Code Page Support

Unicode functionality is supported only on 5250 sessions.

In a Unicode session, the characters in the host source presentation space are translated into Unicode. Attribute bytes are normally translated into blanks.

The XLATE option (that can be specified using the **Set Session Parameters (9)** function) is not supported in a Unicode session. This means that even if this option is issued, the EABs will not be translated to the PC color graphics adapter (CGA) format.

Prerequisite Calls: **Connect Presentation Space (1)**

Call Parameters:

| | Standard Interface | Enhanced Interface |
|-----------------|---|--------------------|
| Function Number | Must be 5 | |
| Data String | Preallocated target Unicode data string. The length (in bytes) should be twice the size (in bytes) of the presentation space. 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 four times the size of the presentation 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:

String containing the Unicode representation of the contents of presentation space is returned

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. |

| Return Code | Explanation |
|-------------|--|
| 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: The following options are supported in a Unicode session for **Copy Presentation Space (5)** and function in the same way as in SBCS:

- NOATTRB
- ATTRB
- NULLATTRB
- EAB
- NOEAB
- NOXLATE
- DISPLAY
- NODISPLAY
- BLANK
- NOBLANK

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

| | Standard Interface | Enhanced Interface |
|-----------------|--------------------|--------------------|
| Function Number | Must be 8 | |

| | Standard Interface | Enhanced Interface |
|-------------|---|--------------------|
| 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 presentation space. | |
| Length | Length of the target data string. | |
| PS Position | Position within the host presentation space of the first byte in your target data string. | |

Return Parameters

This function returns a data string and a return code.

Data String:

Contents of the host 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 following 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 following 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
- EAB/NOEAB
- XLATE/NOXLATE
- BLANK/NOBLANK
- DISPLAY/NODISPLAY
- EAD/NOEAD (for DBCS only)
- NOSO/SPACESO/SO (for DBCS only)
- EXTEND_PS/NOEXTEND_PS

Refer to items 5 on page 149; 13 and 14 on page 152; 15 on page 152; 17 on page 153; and 20 and 21 on page 154

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 left most 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 |

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

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

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

- 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 0 = Off |
| 6 | Blink 1 = On 0 = Off |
| 7 | Reverse 0 = On 1 = Off |

- The following table shows Personal Communications 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 |

| Bit Position | Character Position | Field Attribute Position |
|--------------|---------------------|---|
| 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) |
| 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.

- 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 8 for more information.

Note: 5250 emulation supports a presentation space of 24 rows by 80 columns. In some instances, Communication 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 displays 25th row information on row 24, or on the status bar. For information to be displayed on the status bar, the status bar must be configured. Refer to *Quick Beginnings* for information on configuring the status bar. By the **EXTEND_PS** option, an EHLLAPI application can use the same interface with Communication Manager EHLLAPI and valid presentation space is extended when this condition occurs.

1390/1399 Code Page Support

Unicode functionality is supported only on 3270 and 5250 sessions.

In a Unicode session, the characters in the host source presentation space are translated into Unicode. Attribute bytes are normally translated into blanks.

The XLATE option (that can be specified using the **Set Session Parameters** (9) function) is not supported in a Unicode session. This means that even if this option is issued, the EABs will not be translated to the PC color graphics adapter (CGA) format.

Prerequisite Calls: **Connect Presentation Space** (1)

Call Parameters:

| | Standard Interface | Enhanced Interface |
|-----------------|--|--------------------|
| Function Number | Must be 8 | |
| Data String | Preallocated target Unicode 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 presentation space. | |
| Length | The length of the target Unicode string in Unicode characters. | |
| 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:

String containing the Unicode data is returned

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. |

| Return Code | Explanation |
|-------------|---------------------------------|
| 9 | A system error was encountered. |

Notes on Using This Function: The following options are supported in a Unicode session for **Copy Presentation Space to String** and function in the same way as in DBCS:

- NOATTRB
- ATTRB
- NULLATTRB
- EAB
- NOEAB
- NOXLATE
- DISPLAY
- NODISPLAY
- BLANK
- NOBLANK

1137 Code Page Support

Unicode functionality is supported only on 5250 sessions.

In a Unicode session, the characters in the host source presentation space are translated into Unicode. Attribute bytes are normally translated into blanks.

The XLATE option (that can be specified using the **Set Session Parameters (9)** function) is not supported in a Unicode session. This means that even if this option is issued, the EABs will not be translated to the PC color graphics adapter (CGA) format.

Prerequisite Calls: **Connect Presentation Space (1)**

Call Parameters:

| | Standard Interface | Enhanced Interface |
|-----------------|--|--------------------|
| Function Number | Must be 8 | |
| Data String | Preallocated target data string. The length should be at least twice the number of EBCDIC bytes required to be copied from the presentation space. 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 four times the length of the EBCDIC string that is to be copied from the presentation space. | |
| Length | The length of the target Unicode string in bytes. This length should be at least 2 in a Unicode session. If not, an error code of 2 is returned. | |
| 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: The following options are supported in a Unicode session for **Copy Presentation Space to String** and function in the same way as in SBCS:

- NOATTRB
- ATTRB
- NULLATTRB
- EAB
- NOEAB
- NOXLATE
- DISPLAY
- NODISPLAY
- BLANK
- NOBLANK

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

| | Standard Interface | Enhanced Interface |
|-----------------|--|--------------------|
| Function Number | Must be 33 | |
| Data String | String containing the data to be transferred to a target field in the host presentation space. | |

| | Standard Interface | Enhanced Interface |
|-------------|---|--------------------|
| 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 149; 13 and 14 on page 152; 18 on page 153; and 20 and 21 on page 154 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 147).
- 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, Communication 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 displays 25th row information on row 24, or on the status bar. For information to be displayed on the status bar, the status bar must be configured. Refer to *Quick Beginnings* for information on configuring the status bar. By the **EXTEND_PS** option, an EHLLAPI application can use the same interface with Communication Manager EHLLAPI and valid presentation space is extended when this condition occurs.

1390/1399 Code Page Support

Unicode functionality is supported only on 3270 and 5250 sessions.

STREOT option is not supported in a Unicode session. Please refer to "Set Session Parameters (9)" on page 147 for details.

The XLATE option (that can be specified using the **Set Session Parameters (9)** function) is not supported in a Unicode session. This means that even if this option is issued, the EABs will not be translated to the PC color graphics adapter (CGA) format.

Prerequisite Calls: **Connect Presentation Space (1)**

Call Parameters:

| | Standard Interface | Enhanced Interface |
|-----------------|--|--------------------|
| Function Number | Must be 33 | |
| Data String | String containing the Unicode data to be transferred to a target field in the host presentation space. | |

| | Standard Interface | Enhanced Interface |
|-------------|--|--------------------|
| Length | Length, in number of Unicode characters, of the source Unicode string. Note: The EOT mode is not supported in a Unicode session; therefore, length should be specified for proper functioning of this function in a Unicode session. | |
| 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: The following options are supported in a Unicode session for **Copy String to Field** and function in the same way as in DBCS:

- STRLEN
- EAB
- NOEAB
- NOXLATE
- PUTEAB
- NOPUTEAB

1137 Code Page Support

Unicode functionality is supported only on 5250 sessions.

STREOT option is not supported in a Unicode session. Please refer to “Set Session Parameters (9)” on page 147 for details.

The XLATE option (that can be specified using the **Set Session Parameters (9)** function) is not supported in a Unicode session. This means that even if this option is issued, the EABs will not be translated to the PC color graphics adapter (CGA) format.

Prerequisite Calls: **Connect Presentation Space (1)**

Call Parameters:

| | Standard Interface | Enhanced Interface |
|-----------------|--------------------|--------------------|
| Function Number | Must be 33 | |

| | Standard Interface | Enhanced Interface |
|-------------|--|--------------------|
| Data String | String containing the Unicode data to be transferred to a target field in the host presentation space. | |
| Length | Length, in number of bytes, of the source Unicode string. The length should be at least 2 bytes. If not, an error code of 2 is returned. Note: The EOT mode is not supported in a Unicode session; therefore, length should be specified for proper functioning of this function in a Unicode session. | |
| 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: The following options are supported in a Unicode session for **Copy String to Field** and function in the same way as in SBCS:

- STRLEN
- EAB
- NOEAB
- NOXLATE
- PUTEAB
- NOPUTEAB

Copy String to Presentation Space (15)

| | | |
|-------------|-------------|-----------|
| <i>3270</i> | <i>5250</i> | <i>VT</i> |
| 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

| | Standard Interface | Enhanced Interface |
|-----------------|---|--------------------|
| 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
- EAD/NOEAD (for DBCS only)
- NOSO/SPACESO/SO (for DBCS only)
- EXTEND_PS/NOEXTEND_PS

Refer to items 1 and 2 on page 149; 13 and 14 on page 152; 18 on page 153; and 20 and 21 on page 154 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 147).
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. **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, Communication 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 always displays the same information on the 24th row. By the **EXTEND_PS** option, an EHLLAPI application can use the same interface with Communication Manager EHLLAPI and valid presentation space is extended when this condition occurs.

7. 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.

1390/1399 Code Page Support

Unicode functionality is supported only on 3270 and 5250 sessions.

STREOT option is not supported in a Unicode session. Please refer to "Set Session Parameters (9)" on page 147 for details.

The XLATE option (that can be specified using the **Set Session Parameters (9)** function) is not supported in a Unicode session. This means that even if this option is issued, the EABs will not be translated to the PC color graphics adapter (CGA) format.

Prerequisite Calls: Connect Presentation Space (1)

Call Parameters:

| | Standard Interface | Enhanced Interface |
|-----------------|--|--------------------|
| Function Number | Must be 15 | |
| Data String | String containing the Unicode data to be transferred into the host presentation space. | |
| Length | Length, in number of Unicode characters, of the source Unicode string. Note: The EOT mode is not supported in a Unicode session; therefore, length should be specified for proper functioning of this function in a Unicode session. | |
| 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: The following options are supported in a Unicode session for **Copy String to Presentation Space** and function in the same way as in DBCS:

- STRLEN
- EAB
- NOEAB
- NOXLATE
- PUTEAB
- NOPUTEAB

1137 Code Page Support

Unicode functionality is supported only on 5250 sessions.

STREOT option is not supported in a Unicode session. Please refer to “Set Session Parameters (9)” on page 147 for details.

The XLATE option (that can be specified using the **Set Session Parameters** (9) function) is not supported in a Unicode session. This means that even if this option is issued, the EABs will not be translated to the PC color graphics adapter (CGA) format.

Prerequisite Calls: Connect Presentation Space (1)

Call Parameters:

| | Standard Interface | Enhanced Interface |
|-----------------|---|--------------------|
| Function Number | Must be 15 | |
| Data String | String containing the Unicode data to be transferred into the host presentation space. | |
| Length | Length, in number of Unicode characters, of the source Unicode string. The length should be at least 2 bytes. If not, an error code of 2 is returned. Note: The EOT mode is not supported in a Unicode session; therefore, length should be specified for proper functioning of this function in a Unicode session. | |
| 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: The following options are supported in a Unicode session for **Copy String to Presentation Space** and function in the same way as in SBCS:

- STRLEN
- EAB
- NOEAB
- NOXLATE
- PUTEAB
- NOPUTEAB

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 if a previous **Connect for Structured Fields** was issued.

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

Prerequisite Calls

Connect for Structured Fields (120)

Call Parameters

| | Standard Interface | Enhanced Interface |
|-----------------|-------------------------|--------------------|
| Function Number | Must be 121 | |
| Data String | See the following table | |
| Length | Must be 3 | Must be 8 |
| PS Position | NA | |

Data String Contents

| Byte | | Definition |
|----------|----------|--|
| Standard | Enhanced | |
| 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.

- The **Reset System** (21) function will also free any outstanding asynchronous requests (requests that have not been retrieved by the application using the **Get Request Completion** (125) function).

Disconnect Presentation Space (2)

| | | |
|------|------|-----|
| 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

| | Standard Interface | Enhanced Interface |
|-----------------|--------------------|--------------------|
| 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

- 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).
- Your EHLLAPI application should disconnect from the host presentation space before exiting.
- The **Disconnect Presentation Space** function does not reset the session parameters to the defaults. Your EHLLAPI application must call the **Reset System** (21) function to accomplish this.

Disconnect Window Service (102)

| | | |
|------|------|-----|
| 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

| | Standard Interface | Enhanced Interface |
|-----------------|-------------------------|--------------------|
| Function Number | Must be 102 | |
| Data String | See the following table | |
| Length | 1 | 4 |
| PS Position | NA | |

Data String Contents

| Byte | | Definition |
|----------|----------|--|
| Standard | Enhanced | |
| 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.

EditKey Intercept

This feature enables you to intercept Edit keys in addition to the existing all keystrokes and send them to a session in a Windows 32-bit environment.

Prerequisites

1. Map the Edit functions in the Customize Keyboard window (for example Ctrl+C for edit copy function).
2. Call the Start Keystroke Intercept (50) EHLLAPI function with the call parameter data string value set. The values are as follows:

| Byte Position | Contents |
|---------------|--|
| 1 | One of the following values: <ul style="list-style-type: none"> • A specific host presentation space short name (PSID) • A blank or null indicating a request for the host-connected host presentation space |
| 2 to 4 | Reserved |
| 5 | An option code character: <ul style="list-style-type: none"> • D for AID keystrokes only • L for all keystrokes • E for all keystrokes and Edit keys • M for requesting the asynchronous message mode of the notification (Windows only). If M is specified, a code character D or L, or E must be placed in position 13 |
| 6 to 8 | Reserved |
| 9 to 12 | If M is specified in position 5, the window handle of the window that receives the message. The message is a non-zero return value of RegisterWindowMessage (PCSHLL). |
| 13 | If M is specified in position 5, one of the following values: <ul style="list-style-type: none"> • D for AID keystrokes only • L for all keystrokes • E for all keystrokes and Edit keys |
| 14 to 16 | Reserved |

3. To get the intercepted Edit keys, use the Get Key (51) EHLLAPI function. The key mnemonic returned in the data string for the Edit keys will have M (keystroke type mnemonic) at the 5th byte position. The next 4 bytes will have one of the following Edit key mnemonics based on the Edit key intercepted:

| Key mnemonic | Key intercepted |
|--------------|------------------|
| @W@C | Edit Copy |
| @W@D | Edit Clear |
| @W@E | Edit Copy Append |
| @W@L | Edit Copy Link |
| @W@N | Edit Paste Next |
| @W@V | Edit Paste |
| @W@X | Edit Cut |
| @W@Z | Edit Undo |

4. To send Edit keys to the session, use the Send Key (3) EHLLAPI function. The data string passed as the call parameter can specify the following Edit key mnemonics:

| Key mnemonic | Key sent |
|--------------|-----------|
| @W@C | Edit Copy |

| Key mnemonic | Key sent |
|--------------|------------------|
| @W@D | Edit Clear |
| @W@E | Edit Copy Append |
| @W@L | Edit Copy Link |
| @W@N | Edit Paste Next |
| @W@V | Edit Paste |
| @W@X | Edit Cut |
| @W@Z | Edit Undo |

Notes:

1. You do not have to call the Get Key (51) EHLLAPI function to use the Send Key (3) function. For both Get Key (51) and Send Key (3) functions to handle Edit keys, you must first call Start Keystroke Intercept (50) with the 5th byte position set to E. If the 5th byte contains M, then position 13 must contain E.
2. The expected return values for Start Keystroke Intercept (50), Get Key (51) and Send Key (3) functions have not changed.
3. Any prerequisites from the existing documentation should be followed as well as the prerequisites documented here.

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

| | Standard Interface | Enhanced Interface |
|-----------------|-------------------------|--------------------|
| Function Number | Must be 32 | |
| Data String | See the following table | |
| Length | NA | NA |
| PS Position | See note | |

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 Tb | This field |
| Pb | The previous field, either protected or unprotected. |
| Nb | 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

| | Standard Interface | Enhanced Interface |
|-----------------|-------------------------|--------------------|
| Function Number | Must be 31 | |
| Data String | See the following table | |
| Length | NA | NA |
| PS Position | See note | |

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 Tb | This field |
| Pb | The previous field, either protected or unprotected |
| Nb | 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. |

| Return Code | Explanation |
|-------------|--|
| 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 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

| | Standard Interface | Enhanced Interface |
|-----------------|-------------------------|--------------------|
| Function Number | Must be 124 | |
| Data String | See the following table | |
| Length | Must be 6 | Must be 8 |
| PS Position | NA | |

Data String Contents

| Byte | | Definition |
|----------|----------|---------------------------|
| Standard | Enhanced | |
| 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

| | Standard Interface | Enhanced Interface |
|-----------------|-------------------------|--------------------|
| Function Number | Must be 51 | |
| Data String | See the following table | |
| Length | 8 | 12 |
| PS Position | NA | |

Data String Contents

| Byte | | Definition |
|----------|----------|---|
| Standard | Enhanced | |
| 1 | 1 | One of the following values: <ul style="list-style-type: none"> • 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:

| Byte | | Definition |
|----------|----------|---|
| Standard | Enhanced | |
| 1 | 1 | One of the following values: <ul style="list-style-type: none"> • 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: <ul style="list-style-type: none"> • A for ASCII returned • M for keystroke mnemonic • S for special mnemonic |
| 3–8 | 6–11 | These 6 bytes of the preallocated buffer space are used internally to enqueue and dequeue keystrokes. Possible combinations include: <ul style="list-style-type: none"> • Byte 3 contains an ASCII character and byte 4 contains X'00' • Bytes 3 and 4 contain a double-byte character • Byte 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 8 for more information.

1390/1399 Code Page Support

Unicode functionality is supported only on 3270 and 5250 sessions.

The session option ESC is not supported in a Unicode session; using this option you cannot set a Unicode character as an ESC character. Use the default ESC character @ in a Unicode session. See “Set Session Parameters (9)” on page 147 for details.

Prerequisite Calls: Start Keystroke Intercept (50)

Call Parameters:

| | Standard Interface | Enhanced Interface |
|-----------------|-------------------------|--------------------|
| Function Number | Must be 51 | |
| Data String | See the following table | |
| Length | 8 | 12 |
| PS Position | NA | |

Data String Contents:

| Byte | | Definition |
|----------|----------|---|
| Standard | Enhanced | |
| 1 | 1 | One of the following values: <ul style="list-style-type: none"> • 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 for 32-bit interface:

| Byte | Definition |
|------|--|
| 1 | One of the following values: <ul style="list-style-type: none"> • A 1-character presentation space short name (PSID) • A blank or null indicating a function call for the host-connected presentation |
| 2–4 | Reserved |
| 5 | U is the option code character for a Unicode session. |
| 6–11 | The definition of these bytes is similar to the DBCS session; the only difference is that the Unicode character value is stored in bytes 6 and 7 when the option code character is U. In a DBCS session, the ASCII character value is stored in byte 3 and byte 4 contains 0X'00' when the option code character is A. |

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. |

1137 Code Page Support

Unicode functionality is supported only on 5250 sessions.

The session option ESC is not supported in a Unicode session; using this option you cannot set a Unicode character as an ESC character. Use the default ESC character @ in a Unicode session. See “Set Session Parameters (9)” on page 147 for details.

Prerequisite Calls: **Start Keystroke Intercept** (50)

Call Parameters:

| | Standard Interface | Enhanced Interface |
|-----------------|-------------------------|--------------------|
| Function Number | Must be 51 | |
| Data String | See the following table | |
| Length | 8 | 12 |
| PS Position | NA | |

Data String Contents:

| Byte | | Definition |
|----------|----------|---|
| Standard | Enhanced | |
| 1 | 1 | One of the following values: <ul style="list-style-type: none"> 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 for 32-bit interface:

| Byte | Definition |
|------|--|
| 1 | One of the following values: <ul style="list-style-type: none"> • A 1-character presentation space short name (PSID) • A blank or null indicating a function call for the host-connected presentation |
| 2–4 | Reserved |
| 5 | U is the option code character for a Unicode session. |
| 6–11 | The definition of these bytes is similar to the SBCS session, the only difference is that the Unicode character value is stored in bytes 6 and 7 when the option code character is U. In a DBCS session, the ASCII character value is stored in byte 3 and byte 4 contains 0X'00' when the option code character is A. |

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. |

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 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

Read Structured Fields (126) or Write Structured Fields (127)

Call Parameters

| | Standard Interface | Enhanced Interface |
|-----------------|-------------------------|--------------------|
| Function Number | Must be 125 | |
| Data String | See the following table | |
| Length | Must be 14 | Must be 24 |
| PS Position | NA | |

Data String Contents

| Byte | | Definition |
|----------|----------|--|
| Standard | Enhanced | |
| 1 | 1 | A 1-character presentation space short name (PSID) |
| | 2-4 | Reserved |
| 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 in "Return Parameters" on page 96. 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 in "Return Parameters" on page 96.
- 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 in "Return Parameters" on page 96.

Return Parameters

| Byte | | Definition |
|----------|----------|--|
| Standard | Enhanced | |
| 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. |
| 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 32-bit 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

| | Standard Interface | Enhanced Interface |
|-----------------|-------------------------|--------------------|
| Function Number | Must be 60 | |
| Data String | See the following table | |
| Length | Must be 3 | Must be 8 |
| PS Position | NA | |

Data String Contents

| Byte | | Definition |
|----------|----------|---|
| Standard | Enhanced | |
| 1 | 1 | A 1-character presentation space short name (PSID). |
| | 2-4 | Reserved. |
| 2 | 5 | One of the following characters: <ul style="list-style-type: none"> • L to lock the API. • U to unlock the API. |
| 3 | 6 | One of the following characters: <ul style="list-style-type: none"> • 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. |

| Return Code | Explanation |
|-------------|--|
| 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 32-bit 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

| | Standard Interface | Enhanced Interface |
|-----------------|--------------------------|--------------------|
| Function Number | Must be 61 | |
| Data String | See the following table. | |
| Length | Must be 3 | Must be 8 |
| PS Position | NA | |

Data String Contents

| Byte | | Definition |
|----------|----------|---|
| Standard | Enhanced | |
| 1 | 1 | A 1-character presentation space short name (PSID). |
| | 2-4 | Reserved. |
| 2 | 5 | One of the following characters: <ul style="list-style-type: none"> • L to lock the API. • U to unlock the API. |
| 3 | 6 | One of the following characters: <ul style="list-style-type: none"> • 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. |

| Return Code | Explanation |
|-------------|--|
| 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

There are no prerequisite calls for this function.

Call Parameters

| | Standard Interface | Enhanced Interface |
|-----------------|--------------------|--------------------|
| Function Number | Must be 18 | |

| | Standard Interface | Enhanced Interface |
|-------------|---|--------------------|
| 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 150 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 and the pause value is zero, then the function waits up to 2400 half-second intervals, unless interrupted sooner. If you use the FPAUSE option and the pause value is zero, then the function returns immediately.
4. 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.
5. 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.
6. 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.

Post Intercept Status (52)

| | | |
|------|------|-----|
| 3270 | 5250 | VT |
| Yes | Yes | Yes |

The **Post Intercept Status** function informs the Personal Communications 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

| | Standard Interface | Enhanced Interface |
|-----------------|-------------------------|--------------------|
| 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:

| Byte | | Definition |
|----------|----------|--|
| Standard | Enhanced | |
| 1 | 1 | One of the following values: <ul style="list-style-type: none"> 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: <ul style="list-style-type: none"> 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 Additional Field Attribute (45)

| | | |
|------|------|----|
| 3270 | 5250 | VT |
| No | Yes | No |

The **Query Additional Field Attribute** function returns additional information about the 5250 field containing the input host presentation space position. This information is returned in the data string parameter in the form of a defined structure.

Prerequisite Calls

Connect Presentation Space (1)

Call Parameters

| | Standard Interface | Enhanced Interface |
|-----------------|---|--------------------|
| Function Number | Must be 45. | |
| Data String | 8 bytes long character string. | |
| Length | 8 is implied. | |
| PS Position | Identifies the target. This can be the PS position of any byte within the target field. | |

The calling data string can contain:

| Byte | Definition |
|------|------------|
| 1-8 | Reserved |

Return Parameters

This function returns a data string and a return code.

Data String:

The function returns the following data string.

| Byte | Definition |
|------|--|
| 1-6 | Reserved |
| 7-8 | Two 8-bit unsigned characters that return: <ul style="list-style-type: none"> • R if field is RTL and L if field is LTR. • U if field is upper case and L if field is a normal case field. |

Return Code:

The following return codes are defined:

| Return Code | Explanation |
|-------------|---|
| 0 | The Query Additional 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 | No field was found in this position. |
| 24 | Field is unformatted. |

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

| | Standard Interface | Enhanced Interface |
|-----------------|--------------------------|--------------------|
| Function Number | Must be 42 | |
| Data String | See the following table. | |
| Length | Must be 1 | Must be 4 |
| PS Position | NA | |

The calling data string can contain:

| Byte | | Definition |
|----------|----------|--|
| Standard | Enhanced | |
| 1 | 1 | 1-character short session ID of the host presentation space, or a blank or null indicating request for querying the host-connected session |
| | 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

There are no prerequisite calls for this function.

Call Parameters

| | Standard Interface | Enhanced Interface |
|-----------------|-------------------------|--------------------|
| 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:

| Byte | | Definition |
|----------|----------|---|
| Standard | Enhanced | |
| 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 Communication Event (81)

| | | |
|------|------|-----|
| 3270 | 5250 | VT |
| Yes | Yes | Yes |

The **Query Communication Event** function lets the EHLLAPI program determine whether any communication events have occurred.

Prerequisite Calls

Start Communication Notification (80)

Call Parameters

| | Enhanced Interface |
|-----------------|--|
| Function Number | Must be 81 |
| Data String | 1-character short name of the host presentation space or a blank or null indicating request for updates to the host-connected presentation space |
| Length | 4 is implied |
| PS Position | NA |

The calling data structure contains these elements:

| Byte | Definition |
|------|--|
| 1 | A 1-character presentation space short name (PSID) |
| 2-4 | Reserved |

Return Parameters

| Return Code | Definition |
|-------------|---------------------------------|
| 0 | The function was successful |
| 1 | An incorrect PSID was specified |

| | |
|----|--|
| 8 | No prior call to Start Communication Notification (80) function was called for the PSID |
| 9 | A system error was encountered |
| 21 | The indicated PSID was connected |
| 22 | The Indicated PSID was disconnected |

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

| | Standard Interface | Enhanced Interface |
|-----------------|--------------------|--------------------|
| 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

| | Standard Interface | Enhanced Interface |
|-----------------|---|--------------------|
| 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 left most 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 |

| Bit Position | Meaning |
|--------------|---|
| 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 |
| 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

| | Standard Interface | Enhanced Interface |
|-----------------|---|--------------------|
| 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:

| Byte | | Definition |
|----------|----------|--|
| Standard | Enhanced | |
| 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. |
| 44 | Printing has completed in the printer session. |

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 Session Status (22)

| | | |
|------|------|-----|
| 3270 | 5250 | VT |
| Yes | Yes | Yes |

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

Prerequisite Calls

There are no prerequisite calls for this function.

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: <ol style="list-style-type: none"> 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.

| Byte | | Definition |
|----------|----------|--|
| Standard | Enhanced | |
| 1 | 1 | A 1-character presentation space short name (PSID) |
| | 2-4 | Reserved |
| 2-9 | 5-12 | Session long name (same as profile name; or, if profile not set, same as short name) |
| 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

- 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 8 for more information.

Query Sessions (10)

| | | |
|------|------|-----|
| 3270 | 5250 | VT |
| Yes | Yes | Yes |

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

Prerequisite Calls

There are no prerequisite calls for this function.

Call Parameters

| Function | Description | |
|-----------------|--|--------------------|
| | Standard Interface | Enhanced Interface |
| Function Number | Must be 10 | |
| Data String | Preallocated string of $16n$ bytes long ($12n$ for 16-bit) (n =number of sessions) or more | |
| Length | $12n$ bytes | $16n$ 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 $16n$ bytes long ($12n$ for standard interface), 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 standard interface) session descriptor is as follows:

| Byte | | Definition |
|----------|----------|------------|
| Standard | Enhanced | |

| Byte | | Definition |
|-------|-------|--|
| 1 | 1 | A 1-character presentation space short name (PSID) |
| | 2-4 | Reserved |
| 2-9 | 5-12 | Session long name (same as profile name; or, if profile not set, 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. |
| 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 152 for more information) and by the EXTEND_PS/NOEXTEND_PS option (see item 22 on page 154 for more information).

Notes:

1. When NOCFGZISE 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 Personal Communications 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 in "Return Parameters."

Prerequisite Calls

There are no prerequisite calls for this function.

Call Parameters

| | Standard Interface | Enhanced Interface |
|-----------------|---------------------------------|--------------------|
| 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:

| Byte | | Definition |
|----------|----------|---|
| Standard | Enhanced | |
| 1 | 1 | EHLAPI version number |
| 2-3 | 2-3 | EHLAPI 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 | Personal Communications 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 |
| 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. |

| Return Code | Explanation |
|-------------|--|
| 1 | EHLAPI is not loaded. (PC/3270 only) |
| 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 8 for more information.

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

| | Standard Interface | Enhanced Interface |
|-----------------|---|--------------------|
| Function Number | Must be 103 | |
| 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:

| Byte | | Definition |
|----------|----------|---|
| Standard | Enhanced | |
| 1 | 1 | One of the following values: <ul style="list-style-type: none"> 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.

| Byte | | Definition |
|----------|----------|------------|
| Standard | Enhanced | |
| | | |

| Byte | | Definition |
|-------|-------|--|
| 1 | 1 | One of the following values: <ul style="list-style-type: none"> • 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. |

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

| | Standard Interface | Enhanced Interface |
|-----------------|-------------------------|--------------------|
| Function Number | Must be 126 | |
| Data String | See the following table | |
| Length | 8, 10 or 14 | 20 |
| PS Position | NA | |

The calling data string can contain:

| Byte | | Definition |
|----------|----------|---|
| Standard | Enhanced | |
| 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 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 standard interface) and the function is completed successfully, the following data string is returned:

| Byte | | Definition |
|----------|----------|--|
| Standard | Enhanced | |
| 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:

| 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). The wParam parameter contains Task ID returned by the function call. The 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 standard interface) 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 Single Object**. 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 standard interface) of the data string

- Task ID of asynchronous message mode in positions 17–18 (11–12 for standard interface) 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

There are no prerequisite calls for this function.

Call Parameters

| | Standard Interface | Enhanced Interface |
|-----------------|--|--------------------|
| 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 a single-byte character set (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 iSeries, eServer i5, or System i5 host system:
pc_filename [id:]library file member [option]

Following are examples of the data strings for a double-byte character set (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 iSeries, eServer i5, or System i5 host system:
`pc_filename [id:]library file member [option]`

Note: Parameters within [] are optional. Available options are listed below.

| Host System | Common Options |
|---|---|
| VM/CMS | ASCII, JISCI, CRLF, APPEND, TIME n, CLEAR, NOCLEAR, PROGRESS, QUIET |
| MVS/TSO | ASCII, JISCI, CRLF, APPEND, TIME (n), CLEAR, NOCLEAR, PROGRESS, QUIET, AVBLOCK TRACKS CYLINDERS |
| CICS | ASCII, JISCI, CRLF, NOCRLF, BINARY, TIME n, CLEAR, NOCLEAR, PROGRESS, QUIET |
| i5/OS [®] or OS/400 [®] | ASCII, JISCI, CRLF, APPEND, TIME n, CLEAR, NOCLEAR, PROGRESS, QUIET |
| Note: JISCI 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.

Operating system error codes reported by EHLLAPI are greater than 300. To determine the error code, subtract 300 and refer to the operating system documentation for return codes.

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 149 and items 7 and 8 on page 150 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.

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

| | Standard Interface | Enhanced Interface |
|-----------------|--------------------|--------------------|
| Function Number | Must be 12 | |
| Data String | NA | |
| Length | NA | |
| PS Position | 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

| | Standard Interface | Enhanced Interface |
|-----------------|--------------------|--------------------|
| Function Number | Must be 11 | |
| Data String | NA | |
| Length | NA | |
| PS Position | 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

There are no prerequisite calls for this function.

Call Parameters

| | Standard Interface | Enhanced Interface |
|-----------------|--------------------|--------------------|
| Function Number | Must be 21 | |
| Data String | NA | |
| Length | NA | |
| PS Position | NA | |

Return Parameters

| Return Code | Definition |
|-------------|--|
| 0 | The Reset System function was successful. |
| 1 | EHELLAPI 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 EHELLAPI 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

| | Standard Interface | Enhanced Interface |
|-----------------|--------------------|--------------------|
| Function Number | Must be 30. | |

| | Standard Interface | Enhanced Interface |
|-------------|---|--------------------|
| Data String | Target string for search. | |
| Length | Length of the target data string. Overridden in EOT mode. | |
| PS Position | Identifies the target field. For SRCHALL, 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. | |

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. |
| 24 | The search string was not found, or the host presentation space was unformatted. |

Notes on Using This Function

- 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 148 through 4 on page 149 for more information.
- You can use the **Set Session Parameters** (9) function to determine whether your searches proceed forward (SRCHFRWD) or backward (SRCHBKWD) in a field.
- 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'00C').

Note: 5250 emulation supports a presentation space of 24 rows by 80 columns. In some instances, Communication 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 displays 25th row information on row 24, or on the status bar. For information to be displayed on the status bar, the status bar must be configured. Refer to *Quick Beginnings* for information on configuring the status bar. By the **EXTEND_PS** option, an EHLLAPI application can use the same interface with Communication Manager EHLLAPI and valid presentation space is extended when this condition occurs.

1390/1399 Code Page Support

Unicode functionality is supported only on 3270 and 5250 sessions.

STREOT option is not supported in a Unicode session. Please see "Set Session Parameters (9)" on page 147 for details.

Prerequisite Calls: **Connect Presentation Space (1)**

Call Parameters:

| | Standard Interface | Enhanced Interface |
|-----------------|---|--------------------|
| Function Number | Must be 30. | |
| Data String | Target Unicode string for searching. | |
| Length | Length of the target Unicode string in Unicode characters. Note: The EOT mode is not supported in a Unicode session; therefore, length should be specified for proper functioning of this function in a Unicode session. | |
| PS Position | 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 126. | |

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. |
| 24 | The search string was not found, or the host presentation space was unformatted. |

Notes on Using This Function: The following options are supported in a Unicode session for **Search Field** and function in the same way as in DBCS:

- STRLEN
- SRCHALL
- SRCHFROM
- SRCHFRWD
- SRCHBKWD

1137 Code Page Support

Unicode functionality is supported only on 5250 sessions.

STREOT option is not supported in a Unicode session. Please see “Set Session Parameters (9)” on page 147 for details.

Prerequisite Calls: **Connect Presentation Space (1)**

Call Parameters:

| | Standard Interface | Enhanced Interface |
|-----------------|---|--------------------|
| Function Number | Must be 30. | |
| Data String | Target Unicode string for search. | |
| Length | Length of the target Unicode string in bytes. Note: The EOT mode is not supported in a Unicode session; therefore, length should be specified for proper functioning of this function in a Unicode session. | |
| PS Position | 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 126. | |

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. |
| 24 | The search string was not found, or the host presentation space was unformatted. |

Notes on Using This Function: The following options are supported in a Unicode session for **Search Field** and function in the same way as in SBCS:

- STRLEN
- SRCHALL
- SRCHFROM
- SRCHFRWD
- SRCHBKWD

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

| | Standard Interface | Enhanced Interface |
|-----------------|---|--------------------|
| Function Number | Must be 6. | |
| Data String | Target string for search. | |
| Length | Length of the target data string. Overridden in EOT mode. | |
| PS Position | Position within the host presentation space where the search is to begin (SRCHFRWD option) or to end (SRCHBKWD option). Overridden in SRCHALL (default) mode. | |

Return Parameters

This function returns a length and a return code.

Length:

The following codes are 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

- Four sets of parameters under the **Set Session Parameters** (9) function are related to this function. They are the SRCHALL/SRCHFROM, STRLEN/STREET, SRCHFRWD/SRCHBKWD, and the EOT=c session options. See items 1 on page 148 through 4 on page 149 through for more information.
- 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.
- 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.
- The SRCHFROM option is also useful if you are looking for a keyword that might occur more than once in the host presentation space.
- 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.
- 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'0E00C0F' in the data string is treated as a double-byte character FF (X'00C').

Note: 5250 emulation supports a presentation space of 24 rows by 80 columns. In some instances, Communication 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 displays 25th row information on row 24, or on the status bar. For information to be displayed on the status bar, the status bar must be configured. Refer to *Quick Beginnings* for information on configuring the status bar. By the **EXTEND_PS** option, an EHLLAPI application can use the same interface with Communication Manager EHLLAPI and valid presentation space is extended when this condition occurs.

1390/1399 Code Page Support

Unicode functionality is supported only on 3270 and 5250 sessions.

STREOT option is not supported in a Unicode session. Please refer to "Set Session Parameters (9)" on page 147 for details.

Prerequisite Calls: **Connect Presentation Space (1)**

Call Parameters:

| | Standard Interface | Enhanced Interface |
|-----------------|--|--------------------|
| Function Number | Must be 6. | |
| Data String | Target Unicode string for search. | |
| Length | Length of the target Unicode string in Unicode characters. Note: The EOT mode is not supported in a Unicode session; therefore, length should be specified for proper functioning of this function in a Unicode session. | |
| PS Position | Position within the host presentation space where the search is to begin (SRCHFRWD option) or to end (SRCHBKWD option). Overridden in SRCHALL (default) mode. | |

Return Parameters: This function returns a length and a return code.

Length:

The following codes are 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: The following options are supported in a Unicode session for **Search Presentation Space** (6) and function in the same way as in DBCS:

- STRLEN
- SRCHALL
- SRCHFROM
- SRCHFRWD
- SRCHBKWD

1137 Code Page Support

Unicode functionality is supported only on 5250 sessions.

STREOT option is not supported in a Unicode session. Please refer to “Set Session Parameters (9)” on page 147 for details.

Prerequisite Calls: **Connect Presentation Space** (1)

Call Parameters:

| | Standard Interface | Enhanced Interface |
|-----------------|--|--------------------|
| Function Number | Must be 6. | |
| Data String | Target Unicode string for search. | |
| Length | Length of the target Unicode data string in bytes. Note: The EOT mode is not supported in a Unicode session; therefore, length should be specified for proper functioning of this function in a Unicode session. | |
| PS Position | Position within the host presentation space where the search is to begin (SRCHFRWD option) or to end (SRCHBKWD option). Overridden in SRCHALL (default) mode. | |

Return Parameters: This function returns a length and a return code.

Length:

The following codes are 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: The following options are supported in a Unicode session for **Search Presentation Space** (6) and function in the same way as in SBCS:

- STRLEN
- SRCHALL
- SRCHFROM
- SRCHFRWD
- SRCHBKWD

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

There are no prerequisite calls for this function.

Call Parameters

| | Standard Interface | Enhanced Interface |
|-----------------|---|--------------------|
| Function Number | Must be 90. | |
| Data String | Refer to the examples. | |
| Length | Length of the target data string. Overridden in EOT mode. | |
| PS Position | 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 iSeries, eServer i5, or System i5 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 iSeries, eServer i5, or System i5 host system:

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

Note: Parameters within [] are optional. Available options are listed below. For more information about the options, refer to *Administrator's Guide and Reference*.

| 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 |
| i5/OS or OS/400 | ASCII, JISCII, CRLF, APPEND, SRC, LRECL n, TIME n, CLEAR, NOCLEAR, PROGRESS, QUIET |
| <p>Note:</p> <p>JISCII is valid in a DBCS session for Japan only and ASCII is valid for all other SBCS and DBCS sessions.</p> <p>Note: Time, if specified, overrides the value in Set Session parameters.</p> <p>Note:</p> <p>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.</p> | |

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. |

| Return Code | Explanation |
|-------------|---|
| 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.

Operating system error codes reported by EHLLAPI are greater than 300. To determine the error code, subtract 300 and refer to the operating system documentation for return codes.

Notes on Using This Function

- 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 149 plus items 7 and 8 on page 150 for more information.

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

| | Standard Interface | Enhanced Interface |
|-----------------|--------------------|--------------------|
| Function Number | Must be 3. | |

| | Standard Interface | Enhanced Interface |
|-------------|--|--------------------|
| 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 137. | |
| Length | Length of the source data string. Overridden if in EOT mode. | |
| PS Position | NA | |

Return Parameters

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

Notes on Using This Function

- 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 149, 9 and 10 on page 151, and 19 on page 153 for more information.
- 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.
- If the host is busy, input might be rejected.
- 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.

- 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 137.

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.

- 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 151 (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 ASCII 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 7 shows the mnemonics using uppercase alphabetic characters:

Table 7. Mnemonics with Uppercase Alphabetic Characters

| Mnemonic | Meaning | 3270 | 5250 | 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* | DBCS (Reserved) | Yes | Yes | No |
| @Z | Cursor Right | Yes | Yes | Yes |

Table 8 shows the mnemonics using a number or lowercase alphabetic characters.

Table 8. Mnemonics with Numbers or Lowercase Characters

| Mnemonic | Meaning | 3270 | 5250 | 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 |
| @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 |

Table 9 shows the mnemonics using the combination @A and @alphabetic uppercase (A–Z) key.

Table 9. Mnemonics with @A and @ Uppercase Alphabetic Characters

| Mnemonic | Meaning | 3270 | 5250 | VT |
|----------|-------------|------|------|----|
| @A@C | Test | No | Yes | No |
| @A@D | Word Delete | Yes | Yes | No |
| @A@E | Field Exit | Yes | Yes | No |

Table 9. Mnemonics with @A and @ Uppercase Alphabetic Characters (continued)

| Mnemonic | Meaning | 3270 | 5250 | 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 |

Table 10 shows the mnemonics using the combination @A and @number or @A and @alphabetic lowercase (a-z) key.

Table 10. Mnemonics with @A and @ Lowercase Alphabetic Characters

| Mnemonic | Meaning | 3270 | 5250 | 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 |

Table 11 shows the mnemonics using the combination @A and @special character.

Table 11. Mnemonics with @A and @ Alphanumeric (Special) Characters

| Mnemonic | Meaning | 3270 | 5250 | VT |
|----------|---------------------|------|------|----|
| @A@- | Field - | No | Yes | No |
| @A@+ | Field + | No | Yes | No |
| @A@< | Record Backspace | No | Yes | No |

Table 12 shows the mnemonics using the combination @S , @W, and @alphabetic lowercase.

Table 12. Mnemonics with @S (Shift), @W (Edit) and @ Alphabetic Characters

| Mnemonic | Meaning | 3270 | 5250 | VT |
|----------|--|------|------|-----|
| @S@E | Print Presentation Space on Host | No | Yes | No |
| @S@x | Dup | Yes | Yes | No |
| @S@y | Field Mark | Yes | Yes | No |
| @W@C | Edit Copy | Yes | Yes | Yes |
| @W@D | Edit Clear | Yes | Yes | Yes |
| @W@E | Edit Copy Append | Yes | Yes | Yes |
| @W@L | Edit Copy Link | Yes | Yes | Yes |
| @W@N | Edit Paste Next | Yes | Yes | Yes |
| @W@V | Edit Paste | Yes | Yes | Yes |
| @W@X | Edit Cut | Yes | Yes | Yes |
| @W@Z | Edit Undo | Yes | Yes | Yes |

Note: @W Edit mnemonics are supported only in EHLLAPI functions in Enhanced mode. See Start Keystroke Intercept function under “Summary of EHLLAPI Functions” on page 28.

DBCS Only: Table 13 shows the mnemonics using the combination @X and @number or @alphabetic lowercase (a-z).

Table 13. Mnemonics Using @X and @Alphabetic Lowercase (For DBCS Only)

| Mnemonic | Meaning | 3270 | 5250 | 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: Table 14 on page 141 shows the mnemonics using the combination @M and @number or @alphabetic lowercase (a-z)

Table 14. Mnemonics Using @M, @Q and @Alphabetic Lowercase (For VT Only)

| Mnemonic | Meaning | 3270 | 5250 | VT |
|------------|----------------------------|------|------|-----|
| @M@0 | VT Numeric Pad 0 | No | No | Yes |
| @M@1 | VT Numeric Pad 1 | No | No | Yes |
| @M@2 | VT Numeric Pad 2 | No | No | Yes |
| @M@3 | VT Numeric Pad 3 | No | No | Yes |
| @M@4 | VT Numeric Pad 4 | No | No | Yes |
| @M@5 | VT Numeric Pad 5 | No | No | Yes |
| @M@6 | VT Numeric Pad 6 | No | No | Yes |
| @M@7 | VT Numeric Pad 7 | No | No | Yes |
| @M@8 | VT Numeric Pad 8 | No | No | Yes |
| @M@9 | VT Numeric Pad 9 | No | No | Yes |
| @M@- | VT Numeric Pad - | No | No | Yes |
| @M@, | VT Numeric Pad , | No | No | Yes |
| @M@. | VT Numeric Pad . | No | No | Yes |
| @M@e | VT Numeric Pad Enter | No | No | Yes |
| @M@f | VT Edit Find | No | No | Yes |
| @M@i | VT Edit Insert | No | No | Yes |
| @M@r | VT Edit Remove | No | No | Yes |
| @M@s | VT Edit Select | No | No | Yes |
| @M@p | VT Edit Previous Screen | No | No | Yes |
| @M@n | VT Edit Next Screen | No | No | Yes |
| @M@a | VT PF1 | No | No | Yes |
| @M@b | VT PF2 | No | No | Yes |
| @M@c | VT PF3 | No | No | Yes |
| @M@d | VT PF4 | No | No | Yes |
| @M@h | VT HOld Screen | No | No | Yes |
| @M@(space) | Control Code NUL | No | No | Yes |
| @M@A | Control Code SOH | No | No | Yes |

Table 14. Mnemonics Using @M, @Q and @Alphabetic Lowercase (For VT Only) (continued)

| Mnemonic | Meaning | 3270 | 5250 | VT |
|----------|------------------|------|------|-----|
| @M@B | Control Code STX | No | No | Yes |
| @M@C | Control Code ETX | No | No | Yes |
| @M@D | Control Code EOT | No | No | Yes |
| @M@E | Control Code ENQ | No | No | Yes |
| @M@F | Control Code ACK | No | No | Yes |
| @M@G | Control Code BEL | No | No | Yes |
| @M@H | Control Code BS | No | No | Yes |
| @M@I | Control Code HT | No | No | Yes |
| @M@J | Control Code LF | No | No | Yes |
| @M@K | Control Code VT | No | No | Yes |
| @M@L | Control Code FF | No | No | Yes |
| @M@M | Control Code CR | No | No | Yes |
| @M@N | Control Code SO | No | No | Yes |
| @M@O | Control Code SI | No | No | Yes |
| @M@P | Control Code DLE | No | No | Yes |
| @M@Q | Control Code DC1 | No | No | Yes |
| @M@R | Control Code DC2 | No | No | Yes |
| @M@S | Control Code DC3 | No | No | Yes |
| @M@T | Control Code DC4 | No | No | Yes |
| @M@U | Control Code NAK | No | No | Yes |
| @M@V | Control Code SYN | No | No | Yes |
| @M@W | Control Code ETB | No | No | Yes |
| @M@X | Control Code CAN | No | No | Yes |
| @M@Y | Control Code EM | No | No | Yes |
| @M@Z | Control Code SUB | No | No | Yes |

Table 14. Mnemonics Using @M, @Q and @Alphabetic Lowercase (For VT Only) (continued)

| Mnemonic | Meaning | 3270 | 5250 | VT |
|----------|------------------------|------|------|-----|
| @M@u | Control Code ESC | No | No | Yes |
| @M@v | Control Code FS | No | No | Yes |
| @M@w | Control Code GS | No | No | Yes |
| @M@x | Control Code RS | No | No | Yes |
| @M@y | Control Code US | No | No | Yes |
| @M@z | Control Code DEL | No | No | Yes |
| @Q@A | VT User Defined Key 6 | No | No | Yes |
| @Q@B | VT User Defined Key 7 | No | No | Yes |
| @Q@C | VT User Defined Key 8 | No | No | Yes |
| @Q@D | VT User Defined Key 9 | No | No | Yes |
| @Q@E | VT User Defined Key 10 | No | No | Yes |
| @Q@F | VT User Defined Key 11 | No | No | Yes |
| @Q@G | VT User Defined Key 12 | No | No | Yes |
| @Q@H | VT User Defined Key 13 | No | No | Yes |
| @Q@I | VT User Defined Key 14 | No | No | Yes |
| @Q@J | VT User Defined Key 15 | No | No | Yes |
| @Q@K | VT User Defined Key 16 | No | No | Yes |
| @Q@L | VT User Defined Key 17 | No | No | Yes |
| @Q@M | VT User Defined Key 18 | No | No | Yes |
| @Q@N | VT User Defined Key 19 | No | No | Yes |
| @Q@O | VT User Defined Key 20 | No | No | Yes |
| @Q@a | VT Backtab | No | No | Yes |
| @Q@r | VT Clear Page | No | No | Yes |
| @Q@s | VT Edit | No | No | Yes |

The following table shows the mnemonics using a special character.

Table 15. Mnemonics with Special Character Keys

| Mnemonic | Meaning | 3270 | 5250 | VT |
|----------|--|------|------|-----|
| @@ | @ | Yes | Yes | Yes |
| @\$ | Alternate Cursor (The Presentation Manager Interface only) | Yes | Yes | Yes |
| @< | Backspace | Yes | Yes | Yes |

The following table shows BIDI key mnemonics:

Table 16. BIDI Key Mnemonics

| Mnemonic | Meaning | 3270 | 5250 | VT |
|----------|---------------------------|------|------|----------------------|
| @:@s | Screen Reverse | Yes | Yes | Yes |
| @:@n | Bidi Layer | Yes | Yes | Yes |
| @:@l | Latin Layer | Yes | Yes | Yes |
| @:@F | Field Reverse | Yes | Yes | No |
| @:@p | Push | Yes | No | No |
| @:@e | End Push | Yes | No | No |
| @:@a | Auto Push | Yes | No | No |
| @:@r | Auto Reverse | Yes | No | No |
| @:@d | CSD | Yes | No | No |
| @:@f | Final | Yes | No | No |
| @:@i | Isolated | Yes | No | No |
| @:@m | Middle | Yes | No | No |
| @:@t | Initial | Yes | No | No |
| @:@h | Field Shape | Yes | No | No |
| @:@u | Field Base | Yes | No | No |
| @:@b | Base | No | Yes | No |
| @:@o | Close | No | Yes | No |
| @:@K | Column Heading | No | No | Yes |
| @:@B | Cursor Direction | No | No | Yes |
| @:@D | Encoding Mode | No | No | Yes |
| @:@M | VT Change Display Mode | No | No | Yes (Hebrew only) |

The following character keys are interpreted as they are.

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

1390/1399 Code Page Support

Unicode functionality is supported only on 3270 and 5250 sessions.

STREOT option is not supported in a Unicode session. Please see “Set Session Parameters (9)” on page 147 for details.

The session option ESC is not supported in a Unicode session; using this option, you cannot set a Unicode character as an ESC character. Use the default ESC character @ in a Unicode session. Please see “Set Session Parameters (9)” on page 147 for details.

Prerequisite Calls: Connect Presentation Space (1)

Call Parameters:

| | Standard Interface | Enhanced Interface |
|-----------------|--|--------------------|
| Function Number | Must be 3 | |
| Data String | A Unicode 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 137. | |
| Length | The length of the Unicode string in Unicode characters. | |
| PS Position | NA | |

Return Parameters:

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

Notes on Using This Function: Before sending keystrokes to a PCOMM session, be sure that the session is a Unicode session and the Windows operating system supports Unicode. .

The string length should indicate the number of Unicode characters and not the number of ANSI characters to be sent.

1137 Code Page Support

Unicode functionality is supported only on 5250 sessions.

STREOT option is not supported in a Unicode session. Please see “Set Session Parameters (9)” on page 147 for details.

The session option ESC is not supported in a Unicode session; using this option, you cannot set a Unicode character as an ESC character. Use the default ESC character @ in a Unicode session. Please see “Set Session Parameters (9)” on page 147 for details.

Prerequisite Calls: Connect Presentation Space (1)

Call Parameters:

| | Standard Interface | Enhanced Interface |
|-----------------|--|--------------------|
| Function Number | Must be 3 | |
| Data String | A Unicode 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 137. | |
| Length | Length of the Unicode data string in bytes. If the length is not a multiple of 2 then an error code of 2 is returned. | |
| PS Position | NA | |

Return Parameters:

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

Notes on Using This Function: Before sending keystrokes to a PCOMM session, be sure that the session is a Unicode session. If the session is ANSI and a Unicode string is sent, junk characters will be displayed.

The string length should indicate the number bytes and not the number of Unicode characters to be sent. Therefore the length should be a multiple of 2. If not, a parameter error will be returned by the function.

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

| | Standard Interface | Enhanced Interface |
|-----------------|--------------------|--------------------|
| Function Number | Must be 40 | |
| Data String | NA | |

| | Standard Interface | Enhanced Interface |
|-------------|--|--------------------|
| Length | NA | |
| PS Position | 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.

1137 Code Page Support

The usage of **Set Cursor** in a Unicode session is the same as that for a SBCS session except:

- Unicode functionality is supported only on 5250 sessions.
- In a Unicode session only, if the specified cursor is in the middle of a cluster (for example, a Hindi language cluster), then the cursor is positioned to the beginning of the cluster automatically.

Set Session Parameters (9)

| | | |
|-------------|-------------|-----------|
| <i>3270</i> | <i>5250</i> | <i>VT</i> |
| Yes | Yes | Yes |

The **Set Session Parameters** function lets you change certain default session options in EHLLAPI for all sessions. When EHLLAPI is loaded, the default settings for session options are as indicated by the underscored entries in the tables that appear in “Session Options” on page 148. Any, some, or all of these settings can be changed by including the desired option in the calling data string as explained below. Specified settings remain in effect until:

- Changed by a subsequent **Set Session Parameters** (9) function that specifies a new value.
- The **Reset System** (21) function is executed.
- The EHLLAPI application program is terminated.

The following table lists those EHLLAPI functions that are affected by session options. Functions not listed in the table are not affected by any of the session options. Session options that affect each function are indicated by corresponding entries in the “See Items” column. These entries are indexed to the list that follows “Call Parameters” on page 148.

| Function Number | Function Name | See Items |
|-----------------|-----------------------------------|-------------------------------|
| 1 | Connect Presentation Space | 11, 23, 24 |
| 3 | Send Key | 1, 2, 9, 10, 19 |
| 4 | Wait | 12 |
| 5 | Copy Presentation Space | 5, 13, 14, 15, 17, 20, 21, 22 |
| 6 | Search Presentation Space | 1, 2, 3, 4 |
| 8 | Copy Presentation Space to String | 5, 13, 14, 15, 17, 20, 21, 22 |
| 10 | Query Sessions | 16, 22 |
| 15 | Copy String to Presentation Space | 1, 2, 13, 14, 18, 20, 21, 22 |
| 18 | Pause | 6 |
| 30 | Search Field | 1, 2, 3, 4, 22 |
| 33 | Copy String to Field | 1, 2, 13, 14, 18, 20, 21, 22 |
| 34 | Copy Field to String | 5, 13, 14, 17, 20, 21, 22 |
| 51 | Get Key | 9, 12 |
| 90 | Send File | 1, 2, 7, 8 |
| 91 | Receive File | 1, 2, 7, 8 |
| 101 | Connect Window Services | 23, 24 |

Note: Items 20 and 21 in this table are for DBCS only

Prerequisite Calls

There are no prerequisite calls for this function.

Call Parameters

| | Standard Interface | Enhanced Interface |
|-----------------|---|--------------------|
| Function Number | Must be 9. | |
| Data String | String containing the desired values of those session options that are to be changed. The data string can contain any of the values in the tables of "Session Options." The values should be placed on the data string line, separated by commas or blanks. The sets of parameters are explained in terms of the functions they affect. | |
| Length | Explicit length of the source data string (the STREOT option is not allowed). | |
| PS Position | NA. | |

Session Options

The following tables show the session options. The default is underlined.

- 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 |
|---------------|---|
| <u>STRLEN</u> | An explicit length is passed for all strings. |

| Value | Explanation |
|--------|---|
| 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 |
|----------------|---|
| <u>SRCHALL</u> | 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 |
|-----------------|--|
| <u>SRCHFRWD</u> | 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 |
|----------------|--|
| <u>NOATTRB</u> | 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 |
|---------------|--|
| <u>FPAUSE</u> | 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 |
|----------------|--|
| <u>NOQUIET</u> | SEND and RECEIVE messages are displayed. |
| QUIET | SEND and RECEIVE messages are not displayed. |

8. The statements in the following table determine how long Personal Communications 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 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------|--|-----------|--------------------|---|-----|---|-----|---|-----|---|-----|---|-----|---|-----|---|-----|---|-----|---|-----|---|-----|---|-----|---|-----|---|-----|---|-----|
| <u>TIMEOUT=0</u> | 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 Personal Communications how many 30-second cycles it should accept before issuing a Cancel itself. <table border="1" data-bbox="682 1092 1088 1764"> <thead> <tr> <th>Character</th> <th>Value (in minutes)</th> </tr> </thead> <tbody> <tr><td>1</td><td>0.5</td></tr> <tr><td>2</td><td>1.0</td></tr> <tr><td>3</td><td>1.5</td></tr> <tr><td>4</td><td>2.0</td></tr> <tr><td>5</td><td>2.5</td></tr> <tr><td>6</td><td>3.0</td></tr> <tr><td>7</td><td>3.5</td></tr> <tr><td>8</td><td>4.0</td></tr> <tr><td>9</td><td>4.5</td></tr> <tr><td>J</td><td>5.0</td></tr> <tr><td>K</td><td>5.5</td></tr> <tr><td>L</td><td>6.0</td></tr> <tr><td>M</td><td>6.5</td></tr> <tr><td>N</td><td>7.0</td></tr> </tbody> </table> | 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 |
| 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) command 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). |
| 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). Extended attribute

bytes (EAB) include extended character attributes and extended field attributes.

| Value | Explanation |
|--------------|---|
| <u>NOEAB</u> | Pass data only, no EABs. |
| EAB | <p>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 $2 \times 1920 = 3840$ for a 24-row by 80-column presentation space.</p> <p>Extended attributes for a string of characters may be reported as attributes of the field byte, rather than as attributes of each individual character in the field. In this case, to tell if a particular character or set of characters on a screen is underscored, do a CopyPStoString specifying the position of the field attribute byte (the byte before the field that is displayed on the screen) to get the EAB information that applies to all of the characters in that field.</p> |

Note: When using **EHLLAPI Copy PS to String**, text is copied which should be invisible to the operator. Use the EHLLAPI Set Session Parameters function to set the NODISPLAY option to determine if there is hidden data. This causes EHLLAPI to return nondisplay fields as nulls. Another common procedure for hiding data is to set the foreground and background colors the same (BLACK, for instance) so the text is displayed, but not visible to the human operator. The only way for your application to detect this is to use the EAB and XLATE session parameters and then copying the PS. The foreground/background color of each position is returned and you can determine which characters are invisible.

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)**.

| Value | Explanation |
|----------------|--|
| <u>NOXLATE</u> | EABs are not translated. |
| XLATE | 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 |
|--------------|--------------------------------------|
| <u>BLANK</u> | 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 located in the Personal Communications user-class application data directory:

```
[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 |
|----------------|---|
| <u>CFGSIZE</u> | 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 |
|----------------|---|
| <u>DISPLAY</u> | 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 |
|-----------------|--|
| <u>NOPUTEAB</u> | EAB (or EAD for DBCS) is not contained in the data string of Copy String to Presentation Space or Copy String to Field . |
| 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 Communication Manager/2. For Communication 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 Personal Communications, the data string specified in these functions must consist of character data only even if EAB (or EAD) is valid. But Personal Communications allows that the data string contains EAB (or EAD) by setting PUTEAB to provide the compatibility with Communication 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 |
|--------------|---|
| <u>RETRY</u> | 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 |
|--------------|--|
| <u>NOEAD</u> | 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 |
|-------------|--|
| <u>NOSO</u> | 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, Communication 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 displays 25th row information on row 24, but EHLLAPI normally sees the <i>real</i> 24th row. By EXTEND_PS option, an EHLLAPI application can use the same interface with Communication Manager EHLLAPI and valid presentation space is extended when this condition occurs. |
| <u>NOEXTEND_PS</u> | 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. |
| <u>WRITE_SUPER</u> | The application requires write access and allows only supervisory application to concurrently connect to its presentation space. This is the default value. |

| Value | Explanation |
|-------------|---|
| 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. |

1390/1399 and 1137 Code Page Support

Code page 1390/1399 Unicode functionality is available only for 3270 and 5250 sessions. Code page 1137 Unicode functionality is available only for 5250 sessions.

The following session option differences must be noted for 1390/1399 and 1137 code page support in a Unicode session:

- The session option STREOT should not be used for Unicode strings for the following reasons:
 - The session option STREOT specifies that the length of the string is not explicitly given. An EOT character indicates the end of the string. By scanning

for the EOT character, the length of the string can be found. This EOT character is stored as a single-byte value. The single-byte EOT character cannot be used for Unicode strings.

- *Scenario:* A user sets the EOT character as 'A' whose ASCII value is 0X'41'. If the string buffer that the user passes to the function contains a Unicode character, then the low byte of this Unicode character will be taken as the string delimiter. Therefore, a single-byte EOT character cannot be used as a string delimiter.
- The EOT character cannot be stored as a Unicode character since the **Set Session Parameter** function is independent of the PCOMM session and the same setting applies to all the sessions of PCOMM. If the EOT is to be stored as a Unicode character, then SBCS and DBCS implementations will be affected by the way the EOT character is passed. At present, the EOT character is expected to be a single-byte value.

Note: If you use the session option STREOT, then the results may not be as expected. You can use a single-byte delimiter with the Unicode strings if you are certain that the single-byte delimiter will not be a part of the Unicode values that you are passing in the buffer.

- The session option ESC is not supported in a Unicode session for the same reason as listed for "STREOT" on page 155.
- The session option XLATE is not supported in Unicode. Even if this option is set, it will be ignored.

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

There are no prerequisite calls for this function.

Call Parameters

| Byte | Definition | |
|-----------------|-------------------------|--------------------|
| | Standard Interface | Enhanced Interface |
| 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.

| Byte | | Definition |
|----------|----------|--|
| Standard | Enhanced | |
| 1 | 1 | A 1-character presentation space short name (PSID). |
| | 2-4 | Reserved. |
| 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 | Specify M to request asynchronous message mode (Windows only). |
| | 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 standard interface) and the function is completed successfully, the following data string is returned.

| Byte | | Definition |
|----------|----------|---|
| Standard | Enhanced | |
| 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 standard interface) and the function is completed successfully, the following data string is returned.

| Byte | | Definition |
|----------|----------|--|
| Standard | Enhanced | |
| 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). The wParam parameter will contain the Task ID returned by this function call. The HIWORD of the lParam parameter will contain the Return Code 26, which shows a close intercept occurred, and the LOWORD of the lParam parameter will contain the 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 or semaphore is in a non-signaled state when the start request function returns. 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 session, any subsequent close requests selected by the user from the emulator operations dialog result in a normal stop requested for that session.

Start Communication Notification (80)

| | | |
|------|------|-----|
| 3270 | 5250 | VT |
| Yes | Yes | Yes |

The **Start Communication Notification** function begins the process by which your EHLLAPI application can determine whether the specified session is connected to a host.

After using this function, the application can use **Query Communication Event (81)** to determine whether the session is connected or disconnected.

Prerequisite Calls

There are no prerequisite calls for this function.

Call Parameters

| | Enhanced Interface |
|-----------------|---|
| Function Number | Must be 80 |
| Data String | Preallocated structure; see the following table |
| Length | 16 |
| PSPosition | NA |

The calling data structure contains these elements

| Byte | Definition |
|-------|---|
| 1 | A 1-character presentation space short name (PSID). |
| 2-4 | Reserved |
| 5 | One of the following values: <ul style="list-style-type: none"> • The character C asks for notification when the session either disconnects or connects to the host. • The character A requests the asynchronous mode of notification. When A is specified, position 9-12 returns the address of an event object (Windows). The character C must be placed in position 13. • The character M requests the asynchronous message mode of the notification. When M is specified, the event selection character C must be placed in position 13. |
| 6-8 | Reserved |
| 9-12 | When M is specified in position 5, the window handle of the window that receives the message should be set. The message is a return value of RegisterWindowMessage (PCSHLL)—(not zero). |
| 13 | This should contain the character C if position 5 is A or M. |
| 14-16 | Reserved |

Data String

If A (asynchronous mode) is specified in position 5 of the calling data structure and the function is completed successfully, the following data string is returned:

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

If M (asynchronous message mode) is specified in position 5 of the calling data structure and the function is completed successfully, the following data string is returned:

| Byte | Definition |
|------|--|
| 1 | A 1-character presentation space short-name (PSID) |
| 2-8 | Reserved |
| 9-10 | Task ID of asynchronous message mode |

When the session connects or disconnects an application window receives a message. The message is the return value of RegisterWindow Message (PCSHLL). The wParam contains the Task ID returned by the function call. HIWORD of lParam contains a 21 if the session is connected to the host or a 22 if the session is disconnected. The LOWORD of lParam contains the function number 80.

Return Parameters

| Return Code | Definition |
|-------------|------------|
|-------------|------------|

| | |
|---|---|
| 0 | The function was successful |
| 1 | An incorrect PSID was specified |
| 2 | An error was made in designating parameters |
| 9 | A system error was encountered |

Notes on using this Function

1. An application program can issue this function for multiple host sessions. The **Query Communication Event** (81) function can be used to determine the session communication status.
2. If the application chooses the asynchronous option, it can use the Windows SDK call **WaitForSingleObject** to wait until the sessions communication status has changed.
3. The event object is initially in a non-signaled state. It is signaled each time an event occurs. To receive notification for multiple events the application must put the event object into the non-signaled state each time it is signaled, by using the Windows SDK call **ResetEvent**, or by using function 81 **Query Communications Event**.
4. Multiple calls to this function with the same options from the same application will be ignored.
5. This is not exclusive to one application. Several applications can request this function for the same Session ID.

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

There are no prerequisite calls for this function.

Call Parameters

| | Standard Interface | Enhanced Interface |
|-----------------|--|--------------------|
| Function Number | Must be 23 | |
| Data String | Preallocated string; see the following table | |
| Length | 6 or 7 implied | 16 |
| PS Position | NA | |

The calling data string contains these elements:

| Byte | | Definition |
|----------|----------|------------|
| Standard | Enhanced | |

| Byte | | Definition |
|------|-------|---|
| 1 | 1 | One of the following values: <ul style="list-style-type: none"> • 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: <ul style="list-style-type: none"> • 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). • E The character E asking for notification of completion during a printer session. |
| | 6–8 | Reserved. |
| 3–4 | 9–12 | 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). |
| 7 | 13 | One of the following values if position 5 (2 for 16-bit) is A or M: <ul style="list-style-type: none"> • 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 update. |
| | 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:

| Byte | | Definition |
|----------|----------|------------|
| Standard | Enhanced | |

| Byte | | Definition |
|------|------|--|
| 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 standard interface) and the function is completed successfully, the following data string is returned:

| Byte | | Definition |
|----------|----------|--|
| Standard | Enhanced | |
| 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. An application program can issue this function for multiple 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.
2. If the application chooses the asynchronous option, it can wait for the returned event object or semaphore to determine when a host event has occurred.
3. The event object or semaphore 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.
4. An application cannot request Start Host Notification more than once with the same options.

5. 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

There are no prerequisite calls for this function.

Call Parameters

| | Standard Interface | Enhanced Interface |
|-----------------|---|--------------------|
| Function Number | Must be 50 | |
| Data String | See the following table | |
| Length | Keystroke buffer size EHLLAPI allocates 32 bytes minimum for this buffer. | |
| PS Position | NA | |

The calling data string contains:

| Byte | | Definition |
|----------|----------|--|
| Standard | Enhanced | |
| 1 | 1 | One of the following values: <ul style="list-style-type: none"> • 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. |

| Byte | | Definition |
|------|-------|--|
| 2 | 5 | <p>An option code character:</p> <ul style="list-style-type: none"> • D for AID keystrokes only. • L for all keystrokes. • E for edit keys and all keystrokes (Available in Enhanced mode only) • M for requesting the asynchronous message mode of the notification (Windows only). <p>When M is specified, a code character D, or L, or E (Enhanced Mode) must be placed in position 13 (7 for 16-bit).</p> <p>Prerequisite: keyboard keys must be mapped to edit functions, e.g. Ctrl+C mapped to edit copy function. See Table 12 on page 140 for edit functions supported.</p> |
| | 6–8 | Reserved. |
| 3–4 | 9–12 | 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). |
| 7 | 13 | <p>One of the following values if position 5 (2 for 16-bit) is M:</p> <ul style="list-style-type: none"> • D for AID keystrokes only. • L for all keystrokes. • E for edit keys and all keystrokes. (Available in Enhanced mode only.) |
| | 14–16 | Reserved. |

Data String:

If M (asynchronous message mode) is specified in position 5 (2 for standard interface) and the function is completed successfully, the following data string is returned:

| Byte | | Definition |
|----------|----------|--|
| Standard | Enhanced | |
| 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. |

| Return Code | Explanation |
|-------------|---|
| 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

- 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.

- 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.
- Call the **Stop Keystroke Intercept (53)** function before exiting your EHLLAPI application. Otherwise, keystroke interception remains enabled with unpredictable results.

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 Personal Communications user-class application data directory and no extension should be specified in the function call for the macro name.

Prerequisite Calls

Connect Presentation Space (1)

Call Parameters

| | Standard Interface |
|-----------------|------------------------------|
| Function Number | Must be 110 |
| Data String | See the following table |
| Length | Length of macro name, plus 3 |
| PS Position | NA |

| Byte | | Definition |
|----------|----------|----------------------------|
| Standard | Enhanced | |
| 1-2 | | Reserved |
| 3-n | | 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

| | Standard Interface | Enhanced Interface |
|-----------------|---|--------------------|
| Function Number | Must be 43 | |
| Data String | 1-character short session ID of the host presentation space | |
| Length | 1 | Must be 4 |
| PS Position | NA | |

The calling data string can contain:

| Byte | | Definition |
|----------|----------|--|
| Standard | Enhanced | |
| 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. |

| Return Code | Explanation |
|-------------|--|
| 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 Communication Notification (82)

| | | |
|------|------|-----|
| 3270 | 5250 | VT |
| Yes | Yes | Yes |

The **Stop Communication Notification** function disables the capability of the **Query Communication Event** (81) function to determine whether any communication events have occurred in the specified Session.

Prerequisite Calls

Start Communication Notification (80)

Call Parameters

| | Enhanced Interface |
|-----------------|---|
| Function Number | Must be 82 |
| Data String | 1-character short name of the host presentation space, or a blank or null indicating request for updates to the host-connected presentation space |
| Length | 4 is implied |
| PSPosition | NA |

The calling data structure contains these elements:

| Byte | Definition |
|------|--|
| 1 | A 1-character presentation space short name (PSID) |
| 2-4 | Reserved |

Return Parameters

| Return Code | Definition |
|-------------|--|
| 0 | The function was successful |
| 1 | An incorrect PSID was specified |
| 8 | No prior call to Start Communication Notification (80) function was called for the PSID |
| 9 | A system error was encountered |

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

| | Standard Interface | Enhanced Interface |
|-----------------|------------------------|--------------------|
| 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:

| Byte | | Definition |
|----------|----------|--|
| Standard | Enhanced | |
| 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

| | Standard Interface | Enhanced Interface |
|-----------------|--|--------------------|
| 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:

| Byte | | Definition |
|----------|----------|--|
| Standard | Enhanced | |
| 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

| | Standard Interface | Enhanced Interface |
|-----------------|--------------------|--------------------|
| 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 151.
2. You can use this function to see if the host OIA is inhibited.
3. The **Wait** function is satisfied by the host unlocking the keyboard. Therefore, a return code of 0 does not necessarily mean that the transaction has been completed. To verify completion of the transaction, you should use the **Search Field** (30) function or **Search Presentation Space** (6) function combined with the **Wait** function to look for expected keyword prompts.

Window Status (104)

| | | |
|------|------|-----|
| 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

| | Standard Interface | Enhanced Interface |
|-----------------|-------------------------|--------------------|
| 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:

| Byte | | Definition |
|----------|----------|--|
| Standard | Enhanced | |
| 1 | 1 | A 1-character presentation space short name (PSID) |
| | 2-4 | Reserved |

| Byte | | Definition |
|------|---|--|
| 2 | 5 | A request option value, select one of the following values: <ul style="list-style-type: none"> • 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):

| Byte | | Definition |
|----------|----------|---|
| Standard | Enhanced | |
| 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: <p>X'0001' Change the window size. (Not valid with minimize, maximize, restore, or move.)</p> <p>X'0002' Move the window. (Not valid with minimize, maximize, size, or restore.)</p> <p>X'0004' ZORDER window replacement.</p> <p>X'0008' Set the window to visible.</p> <p>X'0010' Set the window to invisible.</p> <p>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.)</p> <p>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.)</p> <p>X'0400' Set the window to minimized. (Not valid with maximize, restore, size, or move.)</p> <p>X'0800' Set the window to maximized. (Not valid with minimize, restore, size, or move.)</p> <p>X'1000' Restore the window. (Not valid with minimize, maximize, size, or move.)</p> |
| 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.) |

| Byte | | Definition |
|-------|-------|---|
| 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):

| Byte | | Definition |
|----------|----------|---|
| Standard | Enhanced | |
| 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):

| Byte | | Definition |
|----------|----------|------------|
| Standard | Enhanced | |
| | | |

| Byte | | Definition |
|-------|-------|---|
| 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. |
| 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.

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

| | Standard Interface | Enhanced Interface |
|-----------------|-------------------------|--------------------|
| 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:

| Byte | | Definition |
|----------|----------|---|
| Standard | Enhanced | |
| 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. |

| Byte | | Definition |
|-------|-------|--|
| 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 standard interface) and the function is completed successfully, the following data string is returned:

| 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 standard interface) and the function is completed successfully, the following data string is returned:

| 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

- 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.
- 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.
- Return code 37 will be returned if the host is inbound disabled.
- 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.
- 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. |

| Offset | Length | Contents |
|--------|--------|--|
| | | ⋮ |
| 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 standard interface) 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 standard interface) of the data string
- Task ID of asynchronous message mode in position 17–18 (11–12 for standard interface)

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

Chapter 4. WinHLLAPI Extension Functions

This chapter describes the extension functions provided when using WinHLLAPI programming support.

Summary of WinHLLAPI Functions

The following WinHLLAPI functions are available for 3270, 5250, and VT:

- “Wait (4)”
 - “Start Host Notification (23)” on page 180
 - “Start Close Intercept (41)” on page 181
 - “Start Keystroke Intercept (50)” on page 182
 - “Send File (90)” on page 183
 - “Receive File (91)” on page 184
-

WinHLLAPI Asynchronous Functions

The following sections describe the WinHLLAPI asynchronous functions.

WinHLLAPIAsync

This entry point is used for six WinHLLAPI functions that often take a long time to complete. With WinHLLAPIAsync, the function will be launched asynchronously and will not interfere with the continued progression of the calling application. These functions are: **Wait (04)**, **Start Host Notify (23)**, **Start Close Intercept (41)**, **Start Keystroke Intercept (50)**, **Send File (90)**, and **Receive File (91)**, and are described in Chapter 4, “WinHLLAPI Extension Functions.”

HANDLE WinHLLAPIAsync (HWIND hWnd, LPWORD *lpnFunction*, LPBYTE *lpData*, LPWORD *lpnLength*, LPWORD *lpnRetC*)*

The parameter list is the same as WinHLLAPI except a window handle is required before the function number. Since the function operates asynchronously, its completion is signaled by a registered message. The window handle is required as the target of the message.

There are two messages that must be registered by the WinHLLAPI application through calls to **RegisterWindowsMessage()** with the strings **WinHLLAPIAsync** (for all functions except 90 and 91) and **WinHLLAPIAsyncFileTransfer** (for functions 90 and 91). The standard format is as follows:

WPARAM

contains the Task Handle returned by the original function call.

LPARAM

the high word contains the error code and the low word contains the original function number.

Wait (4)

This function determines whether the Host session is in an inhibited state. If, for some reason, the session is in an inhibited state, this function will signal your

application with a message when either the inhibited state expires or your wait period has expired. The amount of time to wait is set with the **Set Session Parameters (9)** function.

Prerequisite Functions: Connect Presentation Space (1)

WinHLLAPIAsync(*hWnd, lpwFunction, lpbyString, lpwLength, lpwReturnCode*)

Call Parameters:

| Parameter | Description |
|--------------------|-------------|
| <i>Data String</i> | NA |
| <i>Data Length</i> | NA |
| <i>PS Position</i> | NA |

Return Codes:

| Code | Description |
|------------------|--|
| WHLLOK | The PS is uninhibited and ready for input. |
| WHLLNOTCONNECTED | Your WinHLLAPI application is not connected to a valid host session. |
| WHLLPSBUSY | Function timed out while still inhibited. |
| WHLLNHIBITED | The PS is inhibited. |
| SHLLSYSERROR | The function failed due to a system error. |
| WHLLCANCEL | The asynchronous function was cancelled. |

Remarks: Asynchronous Wait is used to notify the calling application when the inhibited state of the PS is expired. When inhibited state has expired, this version of **Wait** will post a **WinHLLAPIAsync** message to the window specified by the *hWnd*. The session options **TWAIT**, **LWAIT**, and **NWAIT** affect the length of time that this function will wait. See “Set Session Parameters (9)” on page 147 for details on these session options.

Note: If **NWAIT** is specified in the session parameters and the application registers using revision 1.1 of the WinHLLAPI implementation, the **WinHLLAPIAsync** call will work the same as the **WinHLLAPI** call and not send a message. If revision 1.0 is being used then **Wait** will return a message immediately with the inhibited status of the PS.

Start Host Notification (23)

This function enables you to notify your WinHLLAPI application of changes in the Host Session Presentation Space (PS) or Operation Information Area (OIA).

Prerequisite Functions: There are no prerequisite functions for this function.

WinHLLAPIAsync (*hWnd, lpwFunction, lpbyString, lpwLength, lpwReturnCode*)

Call Parameters:

| Parameter | Description |
|--------------------|---|
| <i>Data String</i> | A 7-byte string in the following format: Byte 1 Short name session ID of the desired Host session, or space or null for the current Host session. Byte 2 Notification mode. "P" for presentation space update only, "O" for OIA update only, "B" for both presentation space and OIA updates. When calling WinHLLAPIAsync, this position can be "A". Byte 3-6 Not used. Provided for compatibility with older applications. Byte 7 Reserved or replaced with one of the following if using WinHLLAPIAsync and A in byte 2: P for presentation space update only, O for OIA update only; and B for both presentation space and OIA updates. |
| <i>Data Length</i> | Length of Host event buffer (256 recommended). |
| <i>PS Position</i> | NA |

Return Parameters:

| Parameter | Description |
|--------------------|---|
| <i>Data String</i> | Same as <i>Data String</i> on the call. |

Return Codes:

| Code | Description |
|--------------------|--|
| WHLLOK | Host notification enabled. |
| WHLLNOTCONNECTED | The specified Host session is invalid. |
| WHLLPARAMETERERROR | One of more parameters are invalid. |
| WHLLSYSERROR | The function failed due to a system error. |
| WHLLCANCEL | The asynchronous function was cancelled. |

Remarks: Once enabled, Host notification is enabled until you call **Stop Host Notification (25)** or **WinHLLAPICancelAsyncRequest()**. The function initiates host notification and immediately returns control to your Windows HLLAPI application. This frees your application to perform other tasks while waiting for host updates. When an update occurs, the function will notify the window specified by *hWnd* with the registered message **WinHLLAPIAsync**.

Start Close Intercept (41)

This function intercepts user requests to close Personal Communications.

Prerequisite Functions: There are no prerequisite functions for this function.

WinHLLAPIAsync (*hWnd*, *lpwFunction*, *lpbyString*, *lpwLength*, *lpwReturnCode*)

Call Parameters:

| Parameter | Description |
|--------------------|---|
| <i>Data String</i> | A 5-byte string for returned semaphore address. The first byte is the session short name of the session to query, or space or null for the current session. |
| <i>Data Length</i> | Must be specified. |
| <i>PS Position</i> | NA |

Return Parameters:

| Parameter | Description |
|--------------------|---|
| <i>Data String</i> | A 5-byte string with the following format: Byte 1 Session short name, or space or null for the current session Bytes 2-5 Semaphore address. |

Return Code:

| Code | Description |
|--------------------|--|
| WHLLOK | The function was successful. |
| WHLLNOTCONNECTED | An invalid presentation space was specified. |
| WHLLPARAMETERERROR | An invalid option was specified. |
| WHLLSYSERROR | The function failed due to a system error. |
| WHLLCANCEL | The asynchronous function was cancelled. |

Remarks: Once enabled, Host notification remains enabled until you call **Stop Close Intercept (43)** or **WinHLLAPICancelAsyncRequest (0)**. Initially, the semaphore is set. After using this function, close requests from the user are discarded and the semaphore is cleared.

The function initiates close intercept and immediately returns control to your Windows HLLAPI application. This frees your application to perform other tasks while waiting for close requests. When a close request occurs, the function will notify the window specified by *hWnd* with the registered message **WinHLLAPIAsync**.

Start Keystroke Intercept (50)

This function intercepts keystrokes sent to a session by the user.

Prerequisite Functions: There are no prerequisite functions for this function.

WinHLLAPIAsync (*hWnd*, *lpwFunction*, *lpbyString*, *lpwLength*, *lpwReturnCode*)

Call Parameters:

| Parameter | Description |
|--------------------|--|
| <i>Data String</i> | A 6-byte string in the following format: Byte 1 Session short name, or space or null for the current Host session. Byte 2 Keystroke intercept code. "D" causes only AID keystrokes to be intercepted; "L" causes all keystrokes to be intercepted. Bytes 3-6 Reserved |
| <i>Data Length</i> | Variable (256 is recommended) |
| <i>PS Position</i> | NA |

Return Code:

| Code | Description |
|--------------------|---|
| WHLLOK | Keystroke intercept has been initiated. |
| WHLLNOTCONNECTED | The Host session presentation space is invalid. |
| WHLLPARAMETERERROR | One or more parameters are invalid. |
| WHLLPSBUSY | Session is busy. |
| WHLLSYSERROR | Function failed due to a system error. |
| WHLLCANCEL | Asynchronous function was cancelled. |

Remarks: The function initiates keystroke intercept and immediately returns control to your Windows HLLAPI application. This frees your application to perform other tasks while waiting for keystrokes. Once initiated, the function will post a **WinHLLAPIAsync** message to the window specified by *hWnd* whenever the user sends a key to the PS. After notification, the intercepted keystrokes can be handled in any way that is allowed by a normal EHLLAPI application. Take note that the keystroke buffer is of limited size so each keystroke should be handled and removed from the buffer.

Send File (90)

This function transfers a file from the PC to the Host.

Prerequisite Functions: There are no prerequisite functions for this function.

WinHLLAPIAsync (*hWnd, lpwFunction, lpbyString, lpwLength, lpwReturnCode*)

Call Parameters:

| Parameter | Description |
|--------------------|---|
| <i>Data String</i> | SEND command parameters. |
| <i>Data Length</i> | Length of <i>Data String</i> . NA if session option EOT is specified. |
| <i>PS Position</i> | NA |

Return Codes:

| Code | Description |
|------------------------|---|
| WHLLOK | File transfer started successfully. |
| WHLLPARAMETERERROR | Parameter error or <i>Data Length</i> is zero or greater than 255. |
| WHLLFTXCOMPLETE | File transfer complete. |
| WHLLFTXSEGMENTED | Transfer is complete with segmented records. |
| WHLLSYSERROR | The function failed due to a system error. |
| WHLLTRANSABORTED | File transfer aborted, either due to the user clicking the cancel button or because the timeout period has elapsed. |
| WHLLFILENOTFOUND | PC file not found. |
| WHLLFTXCOMPLETECICS | File transfer was successful (transfer to CICS). |
| WHLLACCESSDENIED | Access denied to PC file. |
| WHLLMEMORY | Insufficient memory. |
| WHLLINVALIDENVIRONMENT | Invalid environment. |

Remarks: Only one file transfer operation is supported per connected Host session.

The function initiates the file transfer and immediately returns control to your Windows HLLAPI application. This frees your application to perform other tasks while the file transfer is occurring. Once initiated the function will regularly post **WinHLLAPIAsyncFileTransfer** messages to the window specified by *hWnd*. These messages will notify the WinHLLAPI application of the status of the transfer and send a final message when the transfer is complete.

wParm

Is the status indicator: the high byte contains the Session ID, the low byte contains the status. If the low byte is zero, the file transfer is still in progress. If the low byte is one, the file transfer has completed.

IParm If the low byte of *wParm* is zero (in progress), *IParm* is the number of bytes transferred. If the low byte *wParm* is one (completed), *IParm* is the completion code.

Receive File (91)

This function transfers a file from the PC to the Host.

Prerequisite Functions: There are no prerequisite functions for this function.

WinHLLAPIAsync (*hWnd*, *lpwFunction*, *lpbyString*, *lpwLength*, *lpwReturnCode*)

Call Parameters:

| Parameter | Description |
|--------------------|-----------------------------|
| <i>Data String</i> | RECEIVE command parameters. |

| Parameter | Description |
|--------------------|---|
| <i>Data Length</i> | Length of <i>Data String</i> . NA if session option EOT is specified. |
| <i>PS Position</i> | NA |

Return Codes:

| Code | Description |
|------------------------|---|
| WHLLOK | File transfer started successfully. |
| WHLLPARAMETERERROR | Parameter error or <i>Data Length</i> is zero or greater than 255. |
| WHLLFTXCOMPLETE | File transfer complete. |
| WHLLFTXSEGMENTED | Transfer is complete with segmented records. |
| WHLLSYSERROR | The function failed due to a system error. |
| WHLLTRANSABORTED | File transfer aborted, either due to the user clicking the cancel button or because the timeout period has elapsed. |
| WHLLFILENOTFOUND | PC file not found. |
| WHLLFTXCOMPLETECICS | File transfer was successful (transfer to CICS). |
| WHLLACCESSDENIED | Access denied to PC file. |
| WHLLMEMORY | Insufficient memory. |
| WHLLINVALIDENVIRONMENT | Invalid environment. |

Remarks: Only one file transfer operation is supported per connected Host session.

The function initiates the file transfer and immediately returns control to your Windows HLLAPI application. This frees your application to perform other tasks while the file transfer is occurring. Once initiated the function will regularly post **WinHLLAPIAsyncFileTransfer** messages to the window specified by *hWnd*. These messages will notify the WinHLLAPI application of the status of the transfer and send a final message when the transfer is complete.

wParm

Is the status indicator: the high byte contains the Session ID, the low byte contains the status. If the low byte is zero, the file transfer is still in progress. If the low byte is one, the file transfer has completed.

lParm If the low byte of *wParm* is zero (in progress), *lParm* is the number of bytes transferred. If the low byte *wParm* is one (completed), *lParm* is the completion code.

WinHLLAPICancelAsyncRequest

This function cancels an outstanding asynchronous function launched by a call to **WinHLLAPIAsync()**.

Syntax

int WinHLLAPICancelAsyncRequest (HANDLE *hAsyncTask*, WORD *wFunction*)

Parameters

hAsyncTask

The handle returned by WinHLLAPIAsync() when the function was initiated.

wFunction

The function number of the asynchronous task to cancel. Because this parameter is required for revision 1.1 but not in 1.0, it is optional.

With this function, any asynchronous task previously initiated by a call to WinHLLAPIAsync() may be canceled while still outstanding.

Returns

The return value indicates if the specified function was, in fact, canceled. If the function was canceled then the return value is WHLLOK (0). If the outstanding asynchronous function was not cancelled, one of the following codes will be returned.

WHLINVALID

hAsyncTask is not a valid task handle.

WHLALREADY

The asynchronous task specified by *hAsyncTask* has already completed.

Initialization and Termination Functions

The following section describes the initialization and termination functions of WinHLLAPI programming support.

WinHLLAPI Startup

This function is used to register the application with the WinHLLAPI implementation and should be called before any other call to the WinHLLAPI implementation. This implementation supports Versions 1.0 and 1.1 of the WinHLLAPI specification. The WinHLLAPI application should negotiate version compatibility with this function.

Syntax

int WinHLLAPIStartup(WORD *wVersionRequired*, LPWHLLAPIDATA *lpData*)

Parameters

wVersionRequired

This is the version required by the WinHLLAPI application. The low byte contains the major version number and the high byte contains the minor version (or revision) number.

lpData

This is a pointer to a WHLLAPIDATA structure which will receive the implementations version number and a string describing the WinHLLAPI implementation provider. The WHLLAPIDATA structure is defined as:

```
#define WHLLDESCRIPTION_LEN 127
typedef struct tagWHLLAPIDATA
{
    WORD wVersion;
    Char szDescription[WHLLDESCRIPTION_LEN + 1];
}WHLLAPIDATA, * PWHLLAPIDATA, FAR *LPWHLLAPIDATA;
```

Returns

The return value indicates success or failure of registering the WinHLLAPI application with the implementation. If registration was successful, the return value is WHLLOK (zero). Otherwise, it is one of the following:

WHLLSYSNOTREADY

Indicates that the underlying network subsystem is unavailable.

WHLLVERNOTSUPPORTED

Indicates that the version requested is not provided by this implementation. This implementation supports Versions 1.0 and 1.1 only.

WinHLLAPI Cleanup

The WinHLLAPI specification recommends that this function be used by the WinHLLAPI application to de-register from the WinHLLAPI implementation.

Syntax

BOOL WinHLLAPICleanup()

Returns

Returns TRUE if the unregistration was successful. Otherwise, it returns FALSE.

Blocking Routines

The following sections describe the blocking routines supported by WinHLLAPI programming.

Note: Although blocking routines are supported for WinHLLAPI compliance, use of them is not recommended. Use of the WinHLLAPIAsync functions are the recommended method for asynchronous processing.

WinHLLAPIIsBlocking

This function tells the calling WinHLLAPI application thread whether it is in the process of executing a blocking call. A blocking call is any synchronous function that takes a long time to execute and does not return until complete. There are five blocking calls in this implementation of WinHLLAPI. The blocking calls are: **Get Key (51)**, **Wait (4)**, **Pause (18)**, **Send File (90)**, and **Receive File (91)**.

Syntax

BOOL WinHLLAPIIsBlocking()

Returns

If the WinHLLAPI application thread is in the middle of a blocking call, the function returns TRUE, otherwise, it returns FALSE.

Remarks

Because the default blocking-hook allows messages to be processed during blocking calls, it is possible to call the blocking call again.

WinHLLAPISetBlockingHook

This function sets an application-defined procedure to be executed while waiting for the completion of a blocking call. A blocking call is any synchronous function that takes a long time to execute and does not return until complete. There are five blocking calls in this implementation of WinHLLAPI. The blocking calls are: **Get Key (51)**, **Wait (4)**, **Pause (18)**, **Send File (90)**, and **Receive File (91)**.

Syntax

FARPROC WinHLLAPISetBlockingHook(**FARPROC** *lpfnBlockingHook*)

Parameters

lpfnBlockingHook

This is a pointer to the new blocking procedure.

Description

The WinHLLAPI implementation has a default blocking procedure that consists of nothing more than a message handler. This default mechanism is shown in the following example:

```
BOOL DefaultBlockingHook
{
    MSG msg;

    if (PeekMessage (&msg, NULL, 0, 0, xFPM_NOREMOVE))
    {
        if(msg.message == WM_QUIT)
        {
            return FALSE;
        }
        PeekMessage (&msg, NULL, 0, 0, PM_REMOVE);
        TranslateMessage (&msg);
        DispatchMessage (&msg);
    }
    return TRUE;
}
```

The blocking hook is implemented on a per-thread basis. A blocking hook set by this function will stay in effect for the thread until it is replaced by another call to **WinHLLAPISetBlockingHook()** or until the default is restored by a call to **WinHLLAPIUnhookBlockingHook()**.

The Blocking function must return **FALSE** if it receives a **WM_QUIT** message so WinHLLAPI can return control to the application to process the message and terminate gracefully. Otherwise, the function should return **TRUE**.

Returns

This function returns a pointer to the blocking function being replaced.

WinHLLAPIUnhookBlockingHook

This function restores the default blocking-hook for the calling thread.

Syntax

BOOL WinHLLAPIUnhookBlockingHook()

Returns

This function returns **TRUE** if the default blocking mechanism was successfully restored, otherwise it returns **FALSE**.

WinHLLAPICancelBlockingCall

This function cancels an executing blocking call in the *current thread*. A blocking call is any synchronous function that takes a long time to execute and does not return until complete. There are five blocking calls in this implementation of WinHLLAPI. The blocking calls are **Get Key** (51), **Wait** (4), **Pause** (18), **Send File** (90), and **Receive File** (91). If one of these is blocking calls are cancelled, the cancelled function will return **WHLLCANCEL**.

Syntax

int WinHLLAPICancelBlockingCall()

Returns

The return value indicates if the specified function was, in fact, canceled. If the function was canceled, then the return value is WHLLOK (0). If there are no outstanding blocking functions, then the following return code will be returned:

WHLINVALID

Indicates that there is no blocking call currently executing.

Chapter 5. PCSAPI Functions

Personal Communications provides an 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 Personal Communications session itself.

How to Use PCSAPI

You can write application programs using the PCSAPI in C or C++. To develop a PCSAPI application, do the following:

1. Prepare 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.

pcsConnectSession

| | | |
|------|------|-----|
| 3270 | 5250 | VT |
| Yes | Yes | Yes |

The **pcsConnectSession** function starts the communications with a host session specified by the short session ID. The session must already be started. This call is equivalent to the **Communications → Connect** menu item on the emulator session panel.

Function Type

BOOL WINAPI pcsConnectSession(*char cShortSessionID*)

Parameter Type and Description

char cShortSessionID
Presentation space short session ID.

Return Code

| Return Code | Meaning |
|-------------|--|
| TRUE | Function ended successfully. |
| FALSE | It means one of the following things: <ul style="list-style-type: none"> • The session has not started. • An incorrect session ID was specified. • Call failed. |

pcsDisconnectSession

| | | |
|------|------|-----|
| 3270 | 5250 | VT |
| Yes | Yes | Yes |

The **pcsDisconnectSession** function stops the communications link with a host session specified by the short session ID. This only disconnects the link; it does not stop the session. This call is equivalent to the **Communications → Disconnect** menu item on the emulator session panel.

Function Type

BOOL WINAPI pcsDisconnectSession(*char cShortSessionID*)

Parameter Type and Description

char cShortSessionID
Presentation space short session ID.

Return Code

| Return Code | Meaning |
|-------------|--|
| TRUE | Function ended successfully. |
| FALSE | It means one of the following things: <ul style="list-style-type: none"> • The session has not started. • An incorrect session ID was specified. • Call failed. |

pcsQueryConnectionInfo

| | | |
|------|------|----|
| 3270 | 5250 | VT |
| Yes | No | No |

The **pcsQueryConnectionInfo** function returns information about the Telnet connection of the specified host session. The resulting information is returned into the buffer supplied by the application.

Function Type

BOOL WINAPI **pcsQueryConnectionInfo**(*char cShortSessionID*,
*CONNECTIONINFO *ConnectionInfo*)

Parameter Type and Description

char cShortSessionID

Presentation space short session ID.

CONNECTIONINFO *ConnectionInfo

Pointer to a CONNECTIONINFO structure where the connection info data will be returned.

Return Code

| Return Code | Meaning |
|-------------|--|
| TRUE | Function ended successfully. |
| FALSE | It means one of the following things: <ul style="list-style-type: none">• The session has not started.• An incorrect session ID was specified.• The session specified was not a supported connection type for this API (not Telnet). |

ConnectionInfo

The CONNECTIONINFO structure will be filled with the information about the host connection, consisting of the following information:

| Structure | Information |
|----------------------|---|
| Host name | States the name of the currently connected Telnet host. |
| LU name | States the LU name currently assigned. |
| Port number | States the host port number being used for the connection. |
| SSL indicator | Indicates a Secure Connection (1 = secure; 0 = not secure). |

Note: This API is valid only with the 32-bit version of PCSAPI, and only works for Telnet connections.

Example

```
typedef struct_CONNECTIONINFO
{ //Description of a connection @WD06A
  char hostName[63]; //telnet host name @WD06A
  char reserved[1]; //reserved @WD06A
```

```

int portNumber; //host port number @WD06A
char luName[17]; //LU name @WD06A
char reserved2[3]; //reserved @WD06A
BOOL sslIndicator; //Secure Connection @WD06A
indicator
char reserved3[256]; //reserved @WD06A
}CONNECTIONINFO;

```

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 Personal Communications 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 Personal Communications, 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 (this is, the session is connected). |
| PCS_SESSION_API_ENABLED | The session is enabled for programming APIs. If this flag is not set, the EHLAPI 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 supplied by the application.

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: <ul style="list-style-type: none">• The session has not started.• An incorrect session ID was specified. |

pcsSetLinkTimeout

| | | |
|------|------|-----|
| 3270 | 5250 | VT |
| Yes | Yes | Yes |

The **pcsSetLinkTimeout** function sets the idle timeout of a Telnet link which is SSCP owned. This function has no effect on non-TN connections or connections which are not in SSCP owned state. If the timeout value is set to zero the link will not time out. Otherwise the link will time out (disconnect) after being idle in SSCP-owned state for the number of minutes specified.

Function Prototype

ULONG WINAPI **pcsSetLinkTimeout**(*char cShortSessionID, USHORT Timeout*)

Parameter Type and Description

char cShortSessionID

Presentation space short session ID.

USHORT Timeout

Timeout value in minutes. A value of zero disables timeout.

Return Code

| Return Code | Meaning |
|------------------|----------------------------------|
| PCS_SUCCESSFUL | The function ended successfully. |
| PCS_SYSTEM_ERROR | A system error occurred. |

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 complete filename of the profile to load. Path is optional but complete filename must be specified (.ws extension 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. One of the following values from PCSAPI.H:

- PCS_HIDE
- PCS_SHOW
- PCS_MINIMIZE
- PCS_MAXIMIZE

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. |

| Return Code | Meaning |
|-------------|--|
| FALSE | It means one of the following things: <ul style="list-style-type: none"> • The session has not started. • An incorrect session ID was specified. |

Page Setup Functions

The PCSAPI functions listed in this section enable you to control and retrieve the Personal Communications emulator session **Page Setup** settings.

Restrictions

If the following restrictions are not satisfied, the API will fail. The return code indicates the reason for the failure.

- The host session specified in the argument `cShortSessionID` should not be in PDT mode.
- The host session should not be printing when the API is invoked.
- The **File** → **Page Setup** dialog should not be in use.

Some members in the PAGEINFO structure might be valid or supported only for specific session types. If a restriction is not specified, then that member is valid or supported for the following session types:

- 3270 display
- 3270 printer
- 5250 display
- ASCII VT

5250 printer sessions are not supported.

Note: These functions are not currently supported for DBCS and bidirectional sessions.

pcsGetPageSettings

| | | |
|------|------|-----|
| 3270 | 5250 | VT |
| Yes | Yes | Yes |

The **pcsGetPageSettings** function retrieves the host session page settings values (similar to the **File** → **Page Setup** dialog settings). Only the settings in the **Text** tab of the dialog are supported.

Function Type

ULONG WINAPI **pcsGetPageSettings**(*char cShortSessionID, PAGEINFO * const pPageInfo, ULONG * const pErrorInfo*)

Parameter Type and Description

char cShortSessionID

Presentation space short session ID.

PAGEINFO * const pPageInfo

Pointer to PAGEINFO structure, where the page settings are returned.

nFlags Combination of bit flags that indicates which members in the

structure are valid. These flags can be used independently or by ORing them together to restore the property page (defined in PCSAPI32.H). The flags, along with the corresponding valid members in the structure, are as follows:

| Flag | Valid members in the structure |
|--------------------|--------------------------------|
| PCS_PAGE_CPI | nCPI |
| PCS_PAGE_LPI | nLPI |
| PCS_PAGE_FACE_NAME | szFaceName |
| PCS_PAGE_MPL | nMPL |
| PCS_PAGE_MPP | nMPP |

nCPI The number of characters printed per inch.
LOWORD is the actual CPI value.

If Font CPI is configured in the session, HIWORD is 1. If Font CPI is not configured, HIWORD is 0.

nLPI The number of lines printed per inch.
LOWORD is the actual LPI value.

If Font LPI is configured in the session, HIWORD is 1. If Font LPI is not configured, HIWORD is 0.

szFaceName
Face name of the printer font. This must be a null-terminated string.

nFontSize
Size of the printer font.

Note: This is supported only for DBCS host sessions. This is ignored for SBCS host sessions.

nMPL Maximum number of lines that can be printed per page.
This is also called MPL (Maximum Print Lines). Supported range is 1 to 255.

nMPP Maximum number of characters that can be printed per line.
This is also called MPP (Maximum Print Position). Supported range is 1 to 255.

ULONG * const pErrorInfo
Not used. This must be set to NULL by the caller.

Return Code

| Return Code | Value | Meaning |
|-----------------------|-------|--|
| PCS_SUCCESSFUL | 0 | Function ended successfully. |
| PCS_INVALID_ID | 1 | Incorrect session ID was specified. |
| PCS_INVALID_SESS_TYPE | 2 | Not supported for the host session type. |
| PCS_DIALOG_IN_USE | 3 | Failed because the host session Page Setup or Printer Setup dialog was in use. |
| PCS_PRINTING | 4 | Page settings cannot be obtained because host session was printing. |

| Return Code | Value | Meaning |
|------------------|-------|---|
| PCS_PDT_MODE | 5 | Page settings cannot be obtained because host session is in PDT mode. |
| PCS_SYSTEM_ERROR | 9 | A system error occurred. |

Example

```

{
    ULONG Rc = 0;
    PAGEINFO *PageInfo;

    PageInfo = (PAGEINFO *) malloc(sizeof(PAGEINFO));
    memset(PageInfo, 0, sizeof(PAGEINFO));

    PageInfo->nFlags = PCS_PAGE_CPI | PCS_PAGE_LPI | PCS_PAGE_FACE_NAME |
                    PCS_PAGE_MPL | PCS_PAGE_MPP;

    Rc = pcsGetPageSettings('A', PageInfo, NULL);

    if (Rc == PCS_SUCCESSFUL) {
        printf("CPI = %d,
              LPI = %d,
              FaceName = %s,
              MPL = %d,
              MPP = %d\n",
              LOWORD(PageInfo->nCPI),
              LOWORD(PageInfo->nLPI),
              PageInfo->szFaceName,
              PageInfo->nMPL,
              PageInfo->nMPP);

        if (HIWORD(PageInfo->nCPI))
            printf("FontCPI\n");
        else
            printf("No FontCPI\n");

        if (HIWORD(PageInfo->nLPI))
            printf("FontLPI\n");
        else
            printf("No FontLPI\n");
    } else
        printf("Failure. Return code = %d\n", Rc);
    free(PageInfo);
}

```

pcsRestorePageDefaults

| | | |
|------|------|-----|
| 3270 | 5250 | VT |
| Yes | Yes | Yes |

The **pcsRestorePageDefaults** function restores the system default values of the Page Setup property pages defined in the *nFlags* field. This is equivalent to clicking **Default** in the property pages of the **File → Page Setup** dialog. Only the settings in the **Text** tab are supported.

Function Type

ULONG WINAPI **pcsRestorePageDefaults**(*char cShortSessionID, ULONG nFlags*)

Parameter Type and Description

char cShortSessionID

Presentation space short session ID.

ULONG nFlags

The following flag describes the name of the specified **Page Setup** dialog property page. This flag can be bitwise ORed to restore the property page (defined in PCSAPI32.H).

PCS_PAGE_TEXT

This flag describes the Text property page. This is the only property page currently supported.

Return Code

| Return Code | Value | Meaning |
|-----------------------|-------|---|
| PCS_SUCCESSFUL | 0 | Function ended successfully. |
| PCS_INVALID_ID | 1 | Incorrect session ID was specified. |
| PCS_INVALID_SESS_TYPE | 2 | The nFlags parameter has one or more options that are not valid for the host session type. No settings were restored. |
| PCS_DIALOG_IN_USE | 3 | Failed because the host session Page Setup or Printer Setup dialog was in use. |
| PCS_PRINTING | 4 | Page settings cannot be changed because host session was printing. |
| PCS_PDT_MODE | 5 | Page settings cannot be changed because host session is in PDT mode. |
| PCS_SYSTEM_ERROR | 9 | A system error occurred. |

Example

```
{  
    ULONG Rc = 0;  
  
    Rc = pcsRestorePageDefaults('A', PCS_PAGE_TEXT);  
  
    if (Rc != PCS_SUCCESSFUL)  
        printf("Failure. Return code = %d\n", Rc);  
}
```

pcsSetPageSettings

| | | |
|------|------|-----|
| 3270 | 5250 | VT |
| Yes | Yes | Yes |

The **pcsSetPageSettings** function sets the host session page settings. This is similar to configuring the **File → Page Setup** dialog settings. Only the settings in the **Text** tab are supported.

Notes:

1. CPI, LPI, and FontSize are dependent on the FaceName configured in the host session. If this API is used to set CPI, LPI, FontSize, and FaceName together, FaceName is set first, then the dependent properties.

2. If this API is used to set `FaceName` and the dependent properties in separate invocations, set `FaceName` first, then set `CPI`, `LPI` and `FontSize`. Otherwise, each time `FaceName` is set, query `CPI`, `LPI` and `FontSize` and ensure that they have the desired values.
3. If `CPI`, `LPI`, or `FontSize` are set before `FaceName`, then different values for `CPI`, `LPI`, or `FontSize` might be configured in the host session. This might occur if the current `CPI`, `LPI`, or `FontSize` values are not valid for the new `FaceName` set.

Function Type

ULONG WINAPI pcsSetPageSettings(*char cShortSessionID, const PAGEINFO * const pPageInfo, ULONG * const pErrorInfo*)

Parameter Type and Description

char cShortSessionID

Presentation space short session ID.

const PAGEINFO * const pPageInfo

Pointer to `PAGEINFO` structure, where the page settings are mentioned.

nFlags Combination of bit flags that indicates which members in the structure are valid. These flags can be used independently or by ORing them together to restore the property page (defined in `PCSAPI32.H`). The flags, along with the corresponding valid members in the structure, are as follows:

| Flag | Valid members in the structure |
|---------------------------------|--------------------------------|
| <code>PCS_PAGE_CPI</code> | <code>nCPI</code> |
| <code>PCS_PAGE_LPI</code> | <code>nLPI</code> |
| <code>PCS_PAGE_FACE_NAME</code> | <code>szFaceName</code> |
| <code>PCS_PAGE_MPL</code> | <code>nMPL</code> |
| <code>PCS_PAGE_MPP</code> | <code>nMPP</code> |

nCPI The number of characters printed per inch.

To select Font `CPI`, set the `HIWORD` of `nCPI` to 1. `LOWORD` of `nCPI` will be ignored.

To select a particular `CPI` value, do the following:

1. Set the `HIWORD` of `nCPI` to 0.
2. Set the `LOWORD` of `nCPI` to the actual `CPI` value.

nLPI The number of lines printed per inch.

To select Font `LPI`, set the `HIWORD` of `nLPI` to 1. `LOWORD` of `nLPI` will be ignored

To select a particular `LPI` value, do the following:

1. Set the `HIWORD` of `nLPI` to 0.
2. Set the `LOWORD` of `nLPI` to the actual `LPI` value.

szFaceName

Face name of the printer font. This must be a null-terminated string.

nFontSize

Size of the printer font.

Note: This is supported only for DBCS host sessions. This is ignored for SBCS host sessions.

nMPL Maximum number of lines that can be printed per page.

This is also called MPL (Maximum Print Lines). Supported range is 1 to 255.

nMPP Maximum number of characters that can be printed per line.

This is also called MPP (Maximum Print Position). Supported range is 1 to 255.

ULONG * const pErrorInfo

Contains the extended error info when the API fails with the return code of PCS_FAILURE. If the detailed error information is not needed, this flag must be set to NULL by the caller.

This is a combination of bit flags that describe which members of the PAGEINFO structure could not be set successfully. The flags that are defined in PCSAPI32.H are as follows:

| Flag | Valid members in the structure |
|--------------------|---------------------------------------|
| PCS_PAGE_CPI | Only nCPI is not valid. |
| PCS_PAGE_LPI | Only nLPI is not valid. |
| PCS_PAGE_FACE_NAME | Only szFaceName is not valid. |
| PCS_PAGE_MPL | Only nMPL is not valid. |
| PCS_PAGE_MPP | Only nMPP is not valid. |

Return Code

| Return Code | Value | Meaning |
|-----------------------|--------------|---|
| PCS_SUCCESSFUL | 0 | Function ended successfully. |
| PCS_INVALID_ID | 1 | Incorrect session ID was specified. |
| PCS_INVALID_SESS_TYPE | 2 | Not supported for the host session type. |
| PCS_DIALOG_IN_USE | 3 | Failed because the host session Page Setup or Printer Setup dialog was in use. |
| PCS_PRINTING | 4 | Page settings cannot be changed because host session was printing. |
| PCS_PDT_MODE | 5 | Page settings cannot be changed because host session is in PDT mode. |
| PCS_FAILURE | 6 | Host session page settings are not fully applied. This could be because invalid data was given for some or all fields in the PAGEINFO structure. Examine pErrorInfo for details about settings that are not applied. |
| PCS_SYSTEM_ERROR | 9 | A system error occurred. |

Example

```
{
    ULONG Rc = 0, Error = 0;
    PAGEINFO *PageInfo;

    PageInfo = (PAGEINFO *) malloc(sizeof(PAGEINFO));
```

```

memset(PageInfo, 0, sizeof(PAGEINFO));

PageInfo->nFlags = PCS_PAGE_CPI | PCS_PAGE_LPI |
                 PCS_PAGE_FACE_NAME | PCS_PAGE_MPL |
                 PCS_PAGE_MPP;
PageInfo->nCPI = MAKELONG(10, 0);
PageInfo->nLPI = MAKELONG(8, 0);
PageInfo->nMPL = 40;
PageInfo->nMPP = 60;
strcpy(PageInfo->szFaceName, "CourierPS");

Rc = pcsSetPageSettings('A', PageInfo, &Error);

if (Rc != PCS_SUCCESSFUL) {
    printf("Failure. Return code = %d\n", Rc);
    printf("Following members could not be set : ");

    if (Rc == PCS_FAILURE) {
        if (Error & PCS_PAGE_CPI) printf(" nCPI");
        if (Error & PCS_PAGE_LPI) printf(" nLPI");
        if (Error & PCS_PAGE_FACE_NAME) printf(" szFaceName");
        if (Error & PCS_PAGE_MPL) printf(" nMPL");
        if (Error & PCS_PAGE_MPP) printf(" nMPP");
        printf("\n");
    }
}
free(PageInfo);
}

```

Printer Setup Functions

The PCSAPI functions listed in this section enable you to control and retrieve the Personal Communications emulator session **Printer Setup** settings.

Restrictions

If the following restrictions are not met, the API will fail. The return code indicates the reason for the failure.

- The host session should not be printing when the API is invoked.
- The **File** → **Printer Setup** dialog should not be in use.

Note: These functions are not currently supported for DBCS and bidirectional sessions.

pcsGetPrinterSettings

| | | |
|------|------|-----|
| 3270 | 5250 | VT |
| Yes | Yes | Yes |

The **pcsGetPrinterSettings** function retrieves the host session printer settings (similar to the **File** → **Printer Setup** dialog settings).

Function Type

ULONG WINAPI **pcsGetPrinterSettings**(*char cShortSessionID*, *PRINTINFO * const pPrintInfo*, *ULONG * const pErrorInfo*)

Parameter Type and Description

char cShortSessionID

Presentation space short session ID.

PRINTINFO * const pPrintInfo

Pointer to PRINTINFO structure, where the printer settings are specified.

nFlags Must be set to 0. This is ignored.

nBufSize

Size of the buffer allocated for the following fields:

- lpPDTFile
- lpPrtToDskAppFile
- lpPrtToDskSepFile
- lpPrinterName

If more than one of these members is retrieved in a single API call, then the caller must allocate the same size for all the buffers and pass that size in this member.

If this member is set to 0, the fields are ignored. The maximum size required for the buffers of the fields is returned in nSizeNeeded.

nSizeNeeded

The value of this member is determined by conditions related to the following fields:

- lpPDTFile
- lpPrtToDskAppFile
- lpPrtToDskSepFile
- lpPrinterName

The conditions are as follows:

- The value is the number of bytes needed, if the size of the buffer allocated by the caller is not big enough to return the fields listed above.
- The value is the maximum size of the required buffer, if more than one of the fields listed above are obtained by the caller.
- If nBufSize is set to 0 by the caller, this member contains the maximum size required for the buffers of the fields listed above.

bPromptDialog

Possible values are as follows:

- If TRUE, the Printer Setup dialog is shown before printing.
- If FALSE, the Printer Setup dialog is not shown before printing.

bPDTMode

Possible values are as follows:

- If TRUE, the host session is in PDT mode.
- If FALSE, the host session is in non-PDT mode (GDI mode).

lpPDTFile

Must be set to NULL if the caller is not interested in getting this member. The PDT file is returned if this is not a null pointer. This must point to the buffer of size nBufSize allocated by the caller.

When the API returns, this member contains one of the following:

- The fully qualified path name of the session PDT file.
- An empty string ("") if no PDT file is configured in the session.

- A truncated file name if the buffer size is not sufficient. The member `nSizeNeeded` contains the size of the buffer needed.

nPrtMode

This is an enumerated value that indicates the `PrintMode` of the connection. The enum data type `PRINTMODE` is defined in `PCSAPI32.H`. The `nPrtMode` setting must be one of the following:

- **PrtToDskAppend (Print to Disk-Append mode)**
This is equivalent to selecting the **Append** option in the host session **Printer Setup** → **Printer** → **Print to Disk** dialog.
- **PrtToDskSeparate (Print to Disk-Separate mode)**
This is equivalent to selecting the **Separate** option in the host session **Printer Setup** → **Printer** → **Print to Disk** dialog.
- **WinDefaultPrinter (Windows Default Printer mode)**
This is equivalent to selecting the **Use Windows Default Printer** option in the host session **Printer Setup** dialog.
- **SpecificPrinter (Specific Printer mode)**
This is equivalent to selecting a printer in the host session **Printer Setup** dialog, while leaving **Use Windows Default Printer** unchecked.

lpPrtToDskAppFile

Must be set to `NULL` if the caller is not interested in getting this member. The **Print to Disk-Append** file is returned if this is not a null pointer. This must point to the buffer of size `nBufSize` allocated by the caller.

When the API returns, this member contains one of the following:

- The fully qualified path name of the session **Print to Disk-Append** file.
- An empty string ("") if no **Print to Disk-Append** file is configured for the session.
- A truncated file name if the buffer size is not sufficient. The `nSizeNeeded` member contains the size of the buffer needed.

lpPrtToDskSepFile

Must be set to `NULL` if the caller is not interested in getting this member. The **Print to Disk-Separate** file is returned if this is not a null pointer. This must point to the buffer of size `nBufSize` allocated by the caller.

When the API returns, this member contains one of the following:

- The fully qualified path name of the session **Print to Disk-Separate** file.
- An empty string ("") if no **Print to Disk-Separate** file is configured for the session.
- A truncated file name if the buffer size is not sufficient. The `nSizeNeeded` member contains the size of the buffer needed.

lpPrinterName

Must be set to `NULL` if the caller is not interested in getting this member. The name of the printer is returned if this is not a null pointer. This must point to the buffer of size `nBufSize` allocated by the caller.

When the API returns, this member has one of the following:

- The name of the specific printer configured in the session, if the host session nPrtMode is SpecificPrinter.
- The name of the Windows default printer configured in the session, if the host session nPrtMode is WinDefaultPrinter.
- An empty string (""), if the host session nPrtMode is PrtToDskAppend or PrtToDskSeparate.
- A truncated printer name, if the buffer size is not sufficient. nSizeNeeded has the size of the buffer needed.

PrinterName must have the following format:

<Printer name> on <Port Name>

For example:

- IBM InfoPrint 40 PS on Network Port
- HP LaserJet 4050 Series PCL 6 on LPT1

ULONG * const pErrorInfo

This is filled with the extended error info when the API fails with the return code of PCS_FAILURE. pErrorInfo must be set to NULL by the caller, if the details of errors are not needed.

The following section describes the flags that are defined in PCSAPI32.H.

Flags for the pErrorInfo member of the PRINTINFO structure

PCS_PRINT_PRINTMODE_ERROR

PrintMode is not configured in the host session.

PCS_PRINT_PDTFILE_SIZEERR

The buffer size is not sufficient for lpPDTFile, so the file name is truncated. The nSizeNeeded member contains the actual size of the buffer required to return the PDT file.

PCS_PRINT_DSKAPPPFILE_SIZEERR

The buffer size is not sufficient for lpPrtToDskAppFile, so the file name is truncated. The nSizeNeeded member contains the actual size of the buffer required to return the **Print to Disk-Append** file.

PCS_PRINT_DSKSEPFFILE_SIZEERR

The buffer size is not sufficient for lpPrtToDskSepFile, so the file name is truncated. The nSizeNeeded member contains the actual size of the buffer required to return the **Print to Disk-Separate** file.

PCS_PRINT_PRINTERNAME_SIZEERR

The buffer size is not sufficient for lpPrinterName, so the printer name is truncated. The nSizeNeeded member contains the actual size of the buffer required to return the printer name.

Return Code

| Return Code | Value | Meaning |
|--------------------|--------------|---|
| PCS_SUCCESSFUL | 0 | The function ended successfully. |
| PCS_INVALID_ID | 1 | An incorrect session ID was specified. |
| PCS_DIALOG_IN_USE | 3 | Failed because the host session Page Setup or Printer Setup dialog was in use. |
| PCS_PRINTING | 4 | The printer settings could not be changed because the host session was printing. The application must retry later |

| Return Code | Value | Meaning |
|------------------|-------|---|
| PCS_FAILURE | 6 | Some printer settings could not be retrieved successfully. pErrorInfo contains detailed error information on which settings could not be retrieved. |
| PCS_SYSTEM_ERROR | 9 | A system error occurred. |

Example

```

{
    ULONG Rc = 0, Error=0, Size;
    PRINTINFO *PrintInfo;

    PrintInfo = (PRINTINFO *) malloc(sizeof(PRINTINFO));
    memset(PrintInfo, 0, sizeof(PRINTINFO));

    PrintInfo->nBufSize = 0;

    Rc = pcsGetPrinterSettings('A', PrintInfo, &Error);
    if (Rc != PCS_SUCCESSFUL)
        printf("Failure. Return code = %d\n", Rc);
    else {
        Size = PrintInfo->nSizeNeeded;
        PrintInfo->nBufSize = Size;
        PrintInfo->lpPDTFile = (char *)malloc(sizeof(char) * Size);
        PrintInfo->lpPrtToDskAppFile = (char *)malloc(sizeof(char) * Size);
        PrintInfo->lpPrtToDskSepFile = (char *)malloc(sizeof(char) * Size);
        PrintInfo->lpPrinterName = (char *)malloc(sizeof(char) * Size);
        Rc = pcsGetPrinterSettings('A', PrintInfo, &Error);

        if (Rc != PCS_SUCCESSFUL)
            printf("Failure. Return code = %d, Extended Error = 0x%08x\n", Rc, Error);
        else {
            if (PrintInfo->bPromptDialog)
                printf("PromptDialog\n");
            else
                printf("No PromptDialog\n");
            if (PrintInfo->bPDTMode)
                printf("PDT Mode\n");
            else
                printf("Not PDT Mode\n");

            switch(PrintInfo->nPrtMode) {

            case PrtToDskAppend:
                printf("Print to Disk-Append Mode\n");
                break;
            case PrtToDskSeparate:
                printf("Print to Disk-Separate Mode\n");
                break;
            case SpecificPrinter:
                printf("Specific Printer Mode\n");
                break;
            case WinDefaultPrinter:
                printf("Windows Default Printer Mode\n");
                break;
            }
            if (PrintInfo->lpPDTFile[0] == '\0')
                printf("No PDT File configured\n");
            else
                printf("PDT File = %s\n", PrintInfo->lpPDTFile);
            if (PrintInfo->lpPrtToDskAppFile[0] == '\0')
                printf("No Disk Append File configured\n");
            else
                printf("DiskAppend File=%s\n", PrintInfo->lpPrtToDskAppFile);
        }
    }
}

```

```

        if (PrintInfo->lpPrtToDskSepFile[0] == '\\0')
            printf("No Disk Separate File configured\n");
        else
            printf("DiskSeparate File=%s\n", PrintInfo->lpPrtToDskSepFile);
        if ((PrintInfo->nPrtMode == SpecificPrinter) ||
            (PrintInfo->nPrtMode == WinDefaultPrinter))
            printf("Printer = %s\n", PrintInfo->lpPrinterName);
    }
    free(PrintInfo->lpPDTFile);
    free(PrintInfo->lpPrtToDskAppFile);
    free(PrintInfo->lpPrtToDskSepFile);
    free(PrintInfo->lpPrinterName);
}
free(PrintInfo);
}

```

pcsSetPrinterSettings

| | | |
|------|------|-----|
| 3270 | 5250 | VT |
| Yes | Yes | Yes |

The `pcsSetPrinterSettings` function controls the host session printer settings (similar to the **File** → **Printer Setup** dialog settings).

Function Type

ULONG WINAPI `pcsSetPrinterSettings`(*char cShortSessionID, const PRINTINFO * const pPrintInfo, ULONG * const pErrorInfo*)

Parameter Type and Description

char cShortSessionID

Presentation space short session ID.

const PRINTINFO * const pPrintInfo

Pointer to PRINTINFO structure, where the printer settings are mentioned.

nFlags Combination of bit flags that indicates which members in the structure are valid. These flags can be used independently or by ORing them together to restore the property page (defined in PCSAPI32.H). The flags, along with the corresponding valid members in the structure, are as follows:

| Flag | Valid members in the structure |
|-------------------------|---|
| PCS_PRINT_PDT | bPDTMode, lpPDTFile |
| PCS_PRINT_PRINTMODE | nPrtMode, lpPrtToDskAppFile, lpPrtToDskSepFile, lpPrinterName |
| PCS_PRINT_PROMPT_DIALOG | bPromptDialog |

nBufSize

Must be set to 0. This is ignored.

nSizeNeeded

Must be set to 0. This is ignored.

bPromptDialog

Possible values are as follows:

- If TRUE, the Printer Setup dialog is shown before printing.
- If FALSE, the Printer Setup dialog is not shown before printing.

bPDTMode

Possible values are as follows:

- If **TRUE**, the connection is set to PDT mode.
- If **FALSE**, the connection is set to non-PDT mode (GDI mode).

lpPDTFile

Used only if **bPDTMode** is set to **TRUE**. This is ignored if **bPDTMode** is set to **FALSE**.

This is a null-terminated string containing the name of the PDT file and must be one of the following:

- **NULL**
The PDT file that is currently configured in the connection is used. If there is no PDT file already configured in the connection, the API fails with an exception.
- File name, without the path
lpPDTFile in the PDFPDT subfolder in the Personal Communications installation path is used.
- Fully qualified path name of the file
If lpPDTFile does not exist, the API fails.

nPrtMode

This is an enumerated value that indicates the **PrintMode** of the connection. The enum data type **PRINTMODE** is defined in **PCSAPI32.H**. The **nPrtMode** setting must be one of the following:

- **PrtToDskAppend (Print to Disk-Append mode)**
This is equivalent to selecting the **Append** option in the host session **Printer Setup** → **Printer** → **Print to Disk** dialog.
- **PrtToDskSeparate (Print to Disk-Separate mode)**
This is equivalent to selecting the **Separate** option in the host session **Printer Setup** → **Printer** → **Print to Disk** dialog.
- **WinDefaultPrinter (Windows Default Printer mode)**
This is equivalent to selecting the **Use Windows Default Printer** option in the host session **Printer Setup** dialog.
- **SpecificPrinter (Specific Printer mode)**
This is equivalent to selecting a printer in the host session **Printer Setup** dialog, while leaving the **Use Windows Default Printer** option unchecked.

lpPrtToDskAppFile

This is used only if **nPrtMode** is set to **PrtToDskAppend**.

This is a null-terminated string containing the name of the **Print to Disk-Append** file and must be one of the following:

- **NULL**
The file that is currently configured for the **PrtToDskAppend** mode in the connection is used. If there is no PDT file already configured in the connection, the API will fail.
- File name, without the path
The user-class application data directory path is used to locate the file. If the file exists, it is used. Otherwise, it will be created when printing is complete.
- Fully qualified path name of the file

The directory must exist in the path, or the API will fail. It is not necessary that the file exist in the path.

lpPrtToDskSepFile

The possible values are as follows:

- Fully qualified path name of the **Print to Disk-Separate** file for the session.
- An empty string ("") if no **Print to Disk-Separate** file is configured for the session.
- A truncated file name if the buffer size is not sufficient. The `nSizeNeeded` member contains the size of the buffer needed.

lpPrinterName

This is used only if `nPrtMode` is set to `SpecificPrinter`. It is ignored otherwise. This is a null-terminated string containing the printer name. If the printer does not exist, this member fails.

PrinterName must have the following format:

<Printer name> on <Port Name>

For example:

- IBM InfoPrint 40 PS on Network Port
- HP LaserJet 4050 Series PCL 6 on LPT1

ULONG * const pErrorInfo

This is filled with the extended error info when the API fails with the return code of `PCS_FAILURE`. `pErrorInfo` must be set to `NULL` by the caller, if the details of errors are not needed.

The following section describes the flags that are defined in `PCSAPI32.H`.

Flags for the pErrorInfo member of the PRINTINFO structure

PCS_PRINT_PDTMODE_ERROR

This can occur for one of one of the following reasons:

- `bPDTMode` is set to `TRUE`, `lpPDTFile` is set to `NULL`, and there is no PDT file already configured for the host session.
- `nPrtMode` is set to `PrtToDskAppend` or `PrtToDskSeparate`, `PCS_PRINT_PDT` is not set in `nFlags`, and the host session is not already in PDT mode.
- `nPrtMode` is set to `PrtToDskAppend` or `PrtToDskSeparate` and `bPDTMode` is set to `FALSE`.

PCS_PRINT_PDTFILE_ERROR

The file or the path specified in `lpPDTFile` was not found.

PCS_PRINT_PRTTODSK_FILE_ERROR

This can occur for one of one of the following reasons:

- The folder specified in the field `lpPrtToDskAppFile` or `lpPrtToDskSepFile` does not exist or does not have write access.
- An extension is specified in the field `lpPrtToDskSepFile`.

PCS_PRINT_PRINTMODE_ERROR

`nPrtMode` cannot be set successfully. This can occur for one of the following reasons:

- The value of `nPrtMode` is not one of the enumerated constants of the `PRINTMODE` enum data type.

- nPrtMode is set to PrtToDskAppend, lpPrtToDskAppFile is set to NULL, and there is no **Print to Disk-Append** file already configured in the host session.
- nPrtMode is set to PrtToDskSeparate, lpPrtToDskSepFile is set to NULL, and there is no **Print to Disk-Separate** file already configured in the host session.
- nPrtMode is set to SpecificPrinter and the printer given in the lpPrinterName field was not found.
- nPrtMode is set to WinDefaultPrinter and there is no default Windows printer configured in the system.
- bPDTMode is set to FALSE and PCS_PRINT_PRINTMODE is not set in nFlags, but the host session PrintMode is PrtToDskAppend or PrtToDskSeparate.

Return Code

| Return Code | Value | Meaning |
|-------------------|-------|--|
| PCS_SUCCESSFUL | 0 | The function ended successfully. |
| PCS_INVALID_ID | 1 | An incorrect session ID was specified. |
| PCS_DIALOG_IN_USE | 3 | Failed because the host session Page Setup or Printer Setup dialog was in use. |
| PCS_PRINTING | 4 | The printer settings could not be changed because the host session was printing. The application must retry later. |
| PCS_FAILURE | 6 | No host session printer settings were applied. This might occur because invalid data was given for some or all of the fields in the PRINTINFO structure. pErrorInfo contains details about the errors. |
| PCS_SYSTEM_ERROR | 9 | A system error occurred. |

Example

```

{
    ULONG Rc = 0, Error=0;
    PRINTINFO *PrintInfo;
    char PDTFile[] = "epson.pdt";
    char SepFile[] = "DiskSep";

    PrintInfo = (PRINTINFO *) malloc(sizeof(PRINTINFO));
    memset(PrintInfo, 0, sizeof(PRINTINFO));

    PrintInfo->nFlags = PCS_PRINT_PDT | PCS_PRINT_PRINTMODE |
        PCS_PRINT_PROMPT_DIALOG;
    PrintInfo->nBufSize = 0;
    PrintInfo->nSizeNeeded = 0;
    PrintInfo->bPDTMode = TRUE;
    PrintInfo->lpPDTFile =
        (char *)malloc(sizeof(char) * (strlen(PDTFile)+1));
    strcpy(PrintInfo->lpPDTFile, PDTFile);
    PrintInfo->nPrtMode = PrtToDskSeparate;
    PrintInfo->lpPrtToDskSepFile =
        (char *)malloc(sizeof(char) * (strlen(SepFile)+1));
    strcpy(PrintInfo->lpPrtToDskSepFile, SepFile);
    PrintInfo->bPromptDialog = TRUE;
    Rc = pcsSetPrinterSettings('A', PrintInfo, &Error);
    if (Rc != PCS_SUCCESSFUL)
        printf("Failure. Return code = %d, Extended Error = 0x%08x\n", Rc, Error);
}

```

```
    free(PrintInfo->lpPDFFile);  
    free(PrintInfo->lpPrtToDskSepFile);  
    free(PrintInfo);  
}
```

Chapter 6. DDE Functions in a 32-bit Environment

This chapter contains information for DDE functions, as used in a Windows 32-bit environment.

Personal Communications 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. Personal Communications is a DDE server for the open sessions that Personal Communications is 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 7, “Using DDE Functions with a DDE Client Application,” on page 291.

Personal Communications also supports 16-bit DDE applications. See Appendix E, “DDE Functions in a 16-Bit Environment,” on page 357.

Personal Communications DDE Data Items

Microsoft Windows DDE uses a three-level naming scheme to identify data items: application, topic, and item. Table 17 describes these levels.

Table 17. 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 Personal Communications. | 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 <i>Microsoft Windows Software Development Kit Guide to Programming</i> . | PS (presentation space) |

Personal Communications supports IBM327032 and IBM525032 applications as Win32 DDE server.

You can use the following 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 Personal Communications contains these associated data items:

| Item | Function |
|----------|---|
| Formats | Returns the list of clipboard formats (numbers) that Personal Communications is capable of rendering. |
| Status | Returns information about the status of each Personal Communications session. |
| SysCon | Returns the level of Personal Communications support and other system related values. |
| SysItems | Returns the list of data items that are available when connected to the Personal Communications system topic. |
| Topics | Returns the list of Personal Communications 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. |
| CONV | Requests Code Conversion from ASCII to EBCDIC and EBCDIC to ASCII. |
| 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 (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

Table 18 on page 217 lists the DDE functions that are available for use with Personal Communications.

Table 18. DDE Functions Available for Personal Communications

| Function | 3270 | 5250 | VT |
|--|------|------|-----|
| "Code Conversion" on page 218 | Yes | Yes | Yes |
| "Find Field" on page 220 | Yes | Yes | Yes |
| "Get Keystrokes" on page 222 | Yes | Yes | Yes |
| "Get Mouse Input" on page 223 | Yes | Yes | Yes |
| "Get Number of Close Requests" on page 226 | Yes | Yes | Yes |
| "Get Operator Information Area" on page 227 | Yes | Yes | Yes |
| "Get Partial Presentation Space" on page 228 | Yes | Yes | Yes |
| "Get Presentation Space" on page 230 | Yes | Yes | Yes |
| "Get Session Status" on page 232 | Yes | Yes | Yes |
| "Get System Configuration" on page 234 | Yes | Yes | Yes |
| "Get System Formats" on page 235 | Yes | Yes | Yes |
| "Get System Status" on page 236 | Yes | Yes | Yes |
| "Get System SysItems" on page 237 | Yes | Yes | Yes |
| "Get System Topics" on page 238 | Yes | Yes | Yes |
| "Get Trim Rectangle" on page 238 | Yes | Yes | Yes |
| "Initiate Session Conversation" on page 239 | Yes | Yes | Yes |
| "Initiate Structured Field Conversation" on page 240 | Yes | No | No |
| "Initiate System Conversation" on page 240 | Yes | Yes | Yes |
| "Put Data to Presentation Space" on page 241 | Yes | Yes | Yes |
| "Search for String" on page 242 | Yes | Yes | Yes |
| "Send Keystrokes" on page 243 | Yes | Yes | Yes |
| "Session Execute Macro" on page 244 | Yes | Yes | Yes |
| "Set Cursor Position" on page 251 | Yes | Yes | Yes |
| "Set Mouse Intercept Condition" on page 253 | Yes | Yes | Yes |
| "Set Presentation Space Service Condition" on page 255 | Yes | Yes | Yes |
| "Set Session Advise Condition" on page 257 | Yes | Yes | Yes |
| "Set Structured Field Service Condition" on page 258 | Yes | No | No |
| "Start Close Intercept" on page 259 | Yes | Yes | Yes |
| "Start Keystroke Intercept" on page 260 | Yes | Yes | Yes |
| "Start Mouse Input Intercept" on page 261 | Yes | Yes | Yes |
| "Start Read SF" on page 264 | Yes | No | No |
| "Start Session Advise" on page 266 | Yes | Yes | Yes |
| "Stop Close Intercept" on page 267 | Yes | Yes | Yes |
| "Stop Keystroke Intercept" on page 268 | Yes | Yes | Yes |
| "Stop Mouse Input Intercept" on page 268 | Yes | Yes | Yes |
| "Stop Read SF" on page 269 | Yes | No | No |
| "Stop Session Advise" on page 270 | Yes | Yes | Yes |
| "Terminate Session Conversation" on page 270 | Yes | Yes | Yes |
| "Terminate Structured Field Conversation" on page 271 | Yes | No | No |
| "Terminate System Conversation" on page 271 | Yes | Yes | Yes |
| "Write SF" on page 272 | Yes | No | No |

Refer to "Summary of DDE Functions in a Windows 32-Bit Environment" on page 285 for a summary of the DDE functions.

Naming Conventions for Parameters

Most DDE parameter names have local variables. These variables have a prefix that indicates the general type of the parameter, followed by one or more words that describe the content of the parameter. Prefixes presented in this book are:

a Atom

| | |
|-----------|--|
| c | Character (a 1-byte value) |
| f | Bit flags packed into a 16-bit integer |
| h | 16-bit handle |
| p | Short (16-bit) pointer |
| lp | Long (32-bit) pointer |
| w | Short (16-bit) unsigned integer |
| u | Unsigned integer |
| sz | Null-terminated character string |

Code Conversion

| | | |
|------|------|-----|
| 3270 | 5250 | VT |
| Yes | Yes | Yes |

The **Code Conversion** function allows a client application to convert ASCII to EBCDIC or EBCDIC to ASCII. This function is only available to 32-bit applications.

Send the message as follows:

```
PostMessage (hServerWnd,
            WM_DDE_POKE,
            hClientWnd,
            PackDDEIPParam (WM_DDE_POKE, hData, aCONV));
```

where:

hDATA

```
typedef struct tagWCDDE_CONV
{
    BYTE        ddepoke[(sizeof(DDEPOKE)-1)];
    char        szSourceName[256];    // name of memory-mapped file
    char        szTargetName[256];    // name of memory-mapped file
    BYTE        ConvType;             // Conversion method
    WORD        uSourceLength;        // Length of source buffer
    WORD        uTargetLength;        // Length of target buffer
}WCDDE_CONV;

typedef union tagDDE_CONV
{
    DDEPOKE     DDEpoke;
    WCDDE_CONV  DDEConv;
}DDE_CONV;

typedef DDE_CONV FAR *LPDDE_CONV;
```

Conversion Types

```
ConvType = 0x01  ASCII to EBCDIC
ConvType = 0x02  EBCDIC to ASCII
```

Note: The string to be converted must be stored in a memory block that is accessible across processes. In Win32, this can only be accomplished by use of memory-mapped files. The global memory is created and named in the client application and the names are sent to Personal Communications

through the DDE message. The steps required to implement this are demonstrated in the following example:

```
//Steps for a Source Buffer (done in client application)
HANDLE hMapFile;
LPVOID lpMapAddress;
ATOM aCONV;

hMapFile = CreateFileMapping((HANDLE)0xFFFFFFFF, // not a real file
    NULL, // Default security.
    PAGE_READWRITE, // Read/write
    (DWORD)0, // Ignored
    (DWORD)nStringLength, // Length of string
    (LPCTSTR)szSourceName); // Name of
                                // mapping object.

If (hMapFile == NULL)
{
    MessageBox ("Could not create file-mapping Source object.");
    return;
}
// Now treat buffer like local memory
strcpy((LPSTR)lpMapAddress, szConcersionString);

// Repeat steps for a Target Buffer
.....
.....
// Set up ATOM information

aCONV = GlobalAddAtom("CONV"); // MUST be this string

// Post DDE Message Now ....

// When done with memory blocks, clean up
if (!UnmapViewOfFile(lpMapAddress))
{
    MessageBox ("Could not unmap view of Target.");
}

CloseHandle(hMapSFile);

// CODE ENDS
```

Personal Communications Response

The function responds with a WM_DDE_ACK message for DDE_POKE. A result value is returned in the high-order byte of the fsStatus word. The following return codes are valid:

| Return Code | Explanation |
|-------------|---|
| 0x0000 | Normal End |
| 0x0200 | An incorrect conversion type or incorrect parameter was specified |
| 0x0600 | An incorrect format was specified |
| 0x0900 | A system error occurred |
| 0x1000 | The destination buffer was exceeded |
| 0x1100 | An internal translation error occurred |

Find Field

| | | |
|------|------|-----|
| 3270 | 5250 | VT |
| Yes | Yes | Yes |

The **Find Field** function returns information about the specified field to the client. It can be used in two ways.

Send the message as follows:

```
PostMessage( hServerWnd,
             WM_DDE_REQUEST,
             hClientWnd,
             MAKELPARAM(cfFormat, aFIELD) );
```

where:

cfFormat

Identifies the format for the field information. This value can be CF_DSPTEXT or CF_TEXT.

aFIELD

Is the atom that specifies the **Find Field** function. The string identified by the atom can have different values depending on the value of **cfFormat**.

CF_DSPTEXT

If **CF_DSPTEXT** is specified for **cfFormat** then **aFIELD** must be an atom that represents the string, **FIELD**. The PS position must be specified in a previous call to the **Set Presentation Space Service Condition** function. This version will return information only about the field which contains that position. The information will be returned in a **WM_DDE_DATA(hData, aFIELD)** message where:

hData Represents

```
typedef struct tagFINDFIELD
{
    unsigned char
    data[sizeof(DDEDATA)-1];
    unsigned short uFieldStart; //Field start position
    unsigned short uFieldLength; //Field Length
    unsigned char cAttribute; //Attribute character value
    unsigned char ubReserved; //reserved, no information for client
} FINDFIELD;

typedef union tagDDE_FINDFIELD
{
    DDEDATA DDEdata;
    FINDFIELD DDEfield;
} DDE_FINDFIELD, *lpDDE_FINDFIELD;
```

CF_TEXT

If **CF_TEXT** is specified for **cfFormat** then **aFIELD** must be an atom that represents the string, **FIELD (pos, "XX")** where:

pos Is the PS position

XX Is a code representing which field relative to **pos** for which information will be returned. These codes are described below:

| Type | Meaning |
|----------|---|
| bb or Tb | The field containing pos . |
| Pb | The field previous to pos , either protected or unprotected. |
| PP | The previous protected field to pos . |
| PU | The previous unprotected field to pos . |
| Nb | The next field after pos , either protected or unprotected. |
| NP | The next protected field after pos . |
| NU | The next unprotected field after pos . |

Note: The **b** symbol represents a required blank.

These codes must appear in quotes as demonstrated above. The information will be returned in a `WM_DDE_DATA(hData, aFIELD)` message where:

hData Represents

```
typedef struct tagFINDFIELD_CF_TEXT
{
    uchar data[sizeof(DDEDATA)-1];
    uchar Fielddata[80];
} FINDFIELD_CF_TEXT;

typedef FINDFIELD_CF_TEXT FAR *LPFINDFIELD_CF_TEXT;

typedef union tagDDE_FIELD
{
    DDEDATA DDEdata;
    FINDFIELD DDEFindField;
    FINDFIELD_CF_TEXT DDEFindField_cftext;
} DDE_FIELD;

typedef DDE_FIELD FAR *LPDDE_FIELD;
```

Personal Communications Response

If the function is successful, it will respond with a `WM_DDE_DATA` message with information as described above. If it fails, it will return with a `WM_DDE_ACK(wStatus, aFIELD)`. A result value is returned in the low-order byte of the `wStatus` word. The following return codes are valid:

| Return Code | Explanation |
|-------------|------------------------------------|
| 0x0001 | PS position is not valid. |
| 0x0002 | PS is unformatted. |
| 0x0006 | The specified format is not valid. |
| 0x0009 | A system error occurred. |

Structure of the Field Information

The field information will be returned in the `Fielddata` member of the `FINDFIELD_CF_TEXT` structure as a string in the following formats.

For 3270:

"Formatted\t%01d\t%01d\t%01d\t%01d\t%04d\t%04d"

| | | |
|------------|-------------------------------------|-----------|
| FA bit 2 | Unprotected / Protected | 0 or 1 |
| FA bit 3 | Alphanumeric / Numeric | 0 or 1 |
| FA bit 4-5 | Intensity / High / Normal | 1, 2 or 3 |
| FA bit 7 | Unmodified / Modified | 0 or 1 |
| Start Pos | Field Start Position (excluding FA) | |
| Length | Field Length (excluding FA) | |

Note: FA = Field Attribute

For 5250:

"Formatted\t%01d\t%01d\t%01d\t%01d\t%01d\t%01d\t%04d\t%04d"

| | | |
|------------|--|--------|
| FA bit 0 | Field Attribute Flag | 0 or 1 |
| FA bit 1 | Invisible / Visible | 0 or 1 |
| FA bit 2 | Unprotected / Protected | 0 or 1 |
| FA bit 3 | Intensity Low/High | 0 or 1 |
| FA bit 4-6 | Field Type 0 = Alphanumeric 1 = Alphabetic 2 = Numeric Shift 3 = Numeric 4 = Default 5 = Digits only 6 = Mag-Stripe Reader Data 7 = Signed Numeric | 0 — 7 |
| FA bit 7 | Unmodified / Modified | 0 or 1 |
| Start Pos | Field Start Position (excluding FA) | |
| Length | Field Length (excluding FA) | |

Note: FA = Field Attribute

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.

Personal Communications Response

Personal Communications 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 Personal Communications 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

Personal Communications 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.

Personal Communications Response

Personal Communications 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 Personal Communications 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, Personal Communications 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 | |
| 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, Personal Communications 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 |
| 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.

Personal Communications Response

Personal Communications 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 Personal Communications 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

Personal Communications 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.

Personal Communications Response

Personal Communications 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 Personal Communications 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. |

| Return Code | Explanation |
|-------------|--------------------------|
| 9 | A system error occurred. |

Structure of the Operator Information Area

Personal Communications 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 either the PS length or End of Field (EOF) flag by using the **Set Presentation Space Service Condition** function before using this function. If the EOF flag is set to `PCS_EFFECTEOF`, the function will return the entire field specified by the start PS position

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.

Personal Communications Response

Personal Communications 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 Personal Communications 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

Personal Communications returns the part of the presentation space in the format specified in the **Get Partial Presentation Space** request.

If the format is CF_DSPTTEXT, Personal Communications returns the presentation space in the following format:

```
typedef struct tagEPS_CF_DSPTTEXT
{
    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(1pPSFIELDS)
} EPS_CF_DSPTTEXT;

typedef union tagDDE_EPS_CF_DSPTTEXT
{
    DDEDATA      DDEdata;
    EPS_CF_DSPTTEXT DDEeps;
} DDE_EPS_CF_DSPTTEXT, *lpDDE_EPS_CF_DSPTTEXT;

# 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_DSPTTEXT)GlobalLock(hData);
lpps = lpDDE->DDEeps.PSData + lpDDE->DDEeps.uPSOffset;
lpfields = lpDDE->DDEeps.PSData + lpDDE->DDEeps.uFieldOffset;
```

If the format is CF_TEXT, Personal Communications 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]; /* Position 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);
lppps = lpDDE->DDEeps.PS;
lppps_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.

Personal Communications Response

Personal Communications 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 Personal Communications 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

Personal Communications returns the presentation space in the format specified in the **Get Presentation Space** request.

If the format is CF_DSPTEXT, Personal Communications 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(1pPSFIELDS) */
} 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, Personal Communications 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.

Personal Communications Response

Personal Communications 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 Personal Communications 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

Personal Communications 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. |

| Fields | Returned Values | Description |
|-----------------------------------|-----------------|---|
| 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 Personal Communications.

Get System Configuration

| | | |
|------|------|-----|
| 3270 | 5250 | VT |
| Yes | Yes | Yes |

The **Get System Configuration** function returns the level of Personal Communications 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.

Personal Communications Response

Personal Communications 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 Personal Communications 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

Personal Communications 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 Personal Communications |
| Level | NN | The level of Personal Communications |
| 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

| | | |
|-------------|-------------|------------|
| <i>3270</i> | <i>5250</i> | <i>VT</i> |
| <i>Yes</i> | <i>Yes</i> | <i>Yes</i> |

The **Get System Formats** function returns the list of Windows clipboard formats supported by Personal Communications. The client application sends the following message to retrieve the format list supported by Personal Communications:

```
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.

Personal Communications Response

Personal Communications 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 Personal Communications:

- CF_TEXT
- CF_DSPTXT

If Personal Communications 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 Personal Communications 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.

Personal Communications Response

Personal Communications returns the status data item in CF_TEXT format in a DDE DATA message:

```
WM_DDE_DATA(hData, aSTATUS)
```

For each opened session, Personal Communications 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 Personal Communications 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 Systems

| | | |
|------|------|-----|
| 3270 | 5250 | VT |
| Yes | Yes | Yes |

Personal Communications supports the DDE system topic so that a client application can connect to the system topic and retrieve information about Personal Communications and the status of the sessions that Personal Communications is managing.

The **Get System SysItems** function returns the list of data items available in the Personal Communications 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.

Personal Communications Response

Personal Communications 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 Personal Communications:

- SysItems
- Topics
- Status
- Formats
- SysCon

If Personal Communications 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 Personal Communications. 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.

Personal Communications Response

Personal Communications 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 Personal Communications:

- System – System Topic
- SessionA – Session A Topic
- ⋮
- SessionZ – Session Z Topic

Note: 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 Personal Communications 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.

Personal Communications Response

Personal Communications 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 Personal Communications 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 Personal Communications. 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.

Personal Communications Response

If a specific topic is selected and Personal Communications can support a conversation with the client application, Personal Communications acknowledges the INITIATE transaction with:

```
WM_DDE_ACK(aIBM327032, aSessionN)
```

If a topic is not selected (aSessionN = NULL), Personal Communications 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 | No | No |

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 an underscore (_), 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.

Personal Communications Response

If Personal Communications 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,
             PackDDE1Param(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.

Personal Communications Response

Personal Communications 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, Personal Communications 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, string to be searched for, and either the PS Length and Search Direction or End of Field (EOF) flag by using the **Set Presentation Space Service Condition** function before using this function. If the EOF flag is set to PCS_EFFECTEOF, the function will search the entire field specified by the Start PS Position parameter.

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.

Personal Communications Response

Personal Communications 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 Personal Communications 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

Personal Communications 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,
             PackDDE1Param(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.

Personal Communications Response

Personal Communications 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 Personal Communications cannot send the keystrokes to the host, Personal Communications returns a negative ACK message containing one of the following status codes in the low-order byte of the wStatus word. Otherwise, Personal Communications 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 Personal Communications. 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 Personal Communications commands. For a list of commands you can issue, see “Issuing Commands with the Session Execute Macro Function.”

Personal Communications Response

If Personal Communications can process the command string, Personal Communications returns an ACK message containing positive status information to the client. If Personal Communications cannot perform the command string, Personal Communications 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

Any combination of the following file transfer options can be included in *options*: MVS, VM, CICS, QUIET, OS/400, and emulation-specific transfer options, separated by spaces.

Refer to *Administrator's Guide and Reference* 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

Any combination of the following file transfer options can be included in *options*: MVS, VM, CICS, QUIET, OS/400, and emulation-specific transfer options, separated by spaces.

Refer to *Administrator's Guide and Reference* for more information about the transfer options.

SENDKEY Command

[SENDKEY(*token*,*token*)]

Sends keystrokes to Personal Communications, where:

token = *text string*|*command*|*macro macroname*

Notes:

1. Text strings are enclosed in double quotation marks.
2. Macros are prefixed with "macro".
3. The argument string for SENDKEY must be 255 characters or fewer.
4. The following commands are supported.

Table 19. 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 |
| 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 |

Table 19. SENDKEY Command List (continued)

| Command Name | Token | PC/3270 | PC400 | VT |
|----------------------------|---------------|---------|-------|-----|
| 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 | pf1 | Yes | Yes | No |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |
| Program Function Key 5 | pf5 | Yes | Yes | No |

Table 19. SENDKEY Command List (continued)

| Command Name | Token | PC/3270 | PC400 | VT |
|-------------------------|----------------|---------|-------|-----|
| Program Function Key 6 | pf6 | Yes | Yes | Yes |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |
| Program Function Key 20 | pf20 | Yes | Yes | Yes |
| Program Function Key 21 | pf21 | Yes | Yes | No |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |
| Program Function Key 24 | pf24 | Yes | Yes | 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 | view 1 | Yes | Yes | Yes |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |
| Window Relocation 8 | view 8 | X | X | 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 |

Table 19. SENDKEY Command List (continued)

| Command Name | Token | PC/3270 | PC400 | VT |
|-----------------------------|------------------|---------|-------|-----|
| 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 |
| 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,
            PackDDELPParam(WM_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;
```

Personal Communications 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.

Personal Communications Response

Personal Communications receives the cursor information and moves the cursor to the specified position in the PS. If the cursor is positioned successfully, Personal Communications 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). |

| Return Code | Explanation |
|-------------|---|
| 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,
             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 | <p>A string terminated with '\0', consists of the constants defined as follows in any order:</p> <p>L Enable intercepting the left button</p> <p>l Disable intercepting the left button</p> <p>R Enable intercepting the right button</p> <p>r Disable intercepting the right button</p> <p>M Enable intercepting the middle button</p> <p>m Disable intercepting the middle button</p> <p>S Enable intercepting a single click</p> <p>s Disable intercepting a single click</p> <p>D Enable intercepting a double click</p> <p>d Disable intercepting a double click</p> <p>T Retrieve the pointed string</p> <p>t Do not retrieve the pointed string</p> |

If the format is CF_DSPTTEXT, the client program sends the condition in the following structure:

```
typedef struct tagSETMOUSE_CF_DSPTTEXT
{
    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_DSPTTEXT;

typedef union tagDDE_SETMOUSE_CF_DSPTTEXT
{
    DDEPOKE        DDEpoke;
    SETMOUSE_CF_DSPTTEXT DDEcond;
} DDE_SETMOUSE_CF_DSPTTEXT, *lpDDE_SETMOUSE_CF_DSPTTEXT;
```


The following table shows the values in the parameters:

| Parameter Name | Meaning | Value |
|-----------------|---|--|
| bLeftButton | Enable or disable interception of the left mouse button | True Enable intercepting the left button False Disable intercepting the left button |
| bRightButton | Enable or disable interception of the right mouse button | True Enable intercepting the right button False Disable intercepting the right button |
| bMiddleButton | Enable or disable interception of the middle mouse button | True Enable intercepting the middle button False Disable intercepting the middle button |
| bSingleClick | Enable or disable interception of the single click | True Enable intercepting the single click False Disable intercepting the single click |
| bDoubleClick | Enable or disable interception of the double click | True Enable intercepting the double click False Disable intercepting the double click |
| bRetrieveString | Retrieve or do not retrieve the pointed string | True Retrieve the pointed string False Do not retrieve the pointed string |

aMOUSE

Identifies the mouse as the item.

Personal Communications Response

When receiving the **Set Mouse Intercept Condition** request, Personal Communications 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,
             PackDDELPParam(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.
```

The following values are valid for the uEOFflag field:

```
PCS_UNEFFECTEOF 0 /* The PS Area is not truncated at End of Field (EOF).
PCS_EFFECTEOF   1 /* The PS Area is truncated at End of Field (EOF).
```

If the value of uEOFflag is PCS_EFFECTEOF then the PS length and Search Direction are not used.

aEPSCOND

Identifies the item for the **Set Presentation Space Service Condition** function.

Personal Communications Response

If Personal Communications can perform the **Set Presentation Space Service Condition** function, then Personal Communications returns an ACK message:

```
WM_DDE_ACK(wStatus, aEPSCOND)
```

If Personal Communications cannot perform the Set Presentation Space Service Condition function, then Personal Communications 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 266 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.

Personal Communications Response

If Personal Communications can perform the **Set Session Advise Condition** function, Personal Communications returns this ACK message:

```
WM_DDE_ACK(wStatus, aPSCOND)
```

If Personal Communications cannot perform the **Set Session Advise Condition** function, then Personal Communications 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 | No | No |

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,
             PackDDE1Param(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. |
| 4 | Structured Field Service Condition is already set. |
| 6 | The specified format is not valid. |
| 9 | A system error occurred. |

Start Close Intercept

| | | |
|-------------|-------------|------------|
| <i>3270</i> | <i>5250</i> | <i>VT</i> |
| <i>Yes</i> | <i>Yes</i> | <i>Yes</i> |

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 DDE1Param(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.

Personal Communications Response

Personal Communications 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 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 Personal Communications.

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,
             PackDDE1Param(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.

Personal Communications Response

Personal Communications 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 Personal Communications. 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,
            PackDDE1Param(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.

Personal Communications Response

Personal Communications 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, Personal Communications 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 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, Personal Communications 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_DSPTXT;

typedef union tagDDE_MOUSE_CF_DSPTXT
{
    DDEDATA          DDEdata;
    MOUSE_CF_DSPTXT DDEmouse;
} DDE_MOUSE_CF_DSPTXT, *lpDDE_MOUSE_CF_DSPTXT;

```

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 Personal Communications.

Start Read SF

| | | |
|------|------|----|
| 3270 | 5250 | VT |
| Yes | No | No |

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,
            PackDDEIParam(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_DSPTXT DDEmouse;
} DDE_MOUSE_CF_DSPTXT, *lpDDE_MOUSE_CF_DSPTXT;
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 Personal Communications 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 257.

The client sends the following command to begin a session advise.

```

PostMessage( hServerWnd,
             WM_DDE_ADVISE,
             hClientWnd,
             PackDDE1Param(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.

Personal Communications Response

Personal Communications 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 Personal Communications. The format of the data item is the same as if the client requested the data item via a `DDE_REQUEST`.

Stop Close Intercept

| | | |
|-------------|-------------|------------|
| <i>3270</i> | <i>5250</i> | <i>VT</i> |
| <i>Yes</i> | <i>Yes</i> | <i>Yes</i> |

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.

Personal Communications Response

If Personal Communications can perform the `DDE_UNADVISE`, Personal Communications returns an ACK message containing positive status information to the client:

`WM_DDE_ACK(wStatus, aCLOSE)`

If Personal Communications cannot perform the DDE_UNADVISE, Personal Communications 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.

Personal Communications Response

If Personal Communications can perform the DDE_UNADVISE, Personal Communications returns an ACK message containing positive status information to the client:

```
WM_DDE_ACK(wStatus, aKEYS)
```

If Personal Communications cannot perform the DDE_UNADVISE, Personal Communications 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.

Personal Communications Response

If Personal Communications can perform the DDE_UNADVISE, Personal Communications returns an ACK message containing positive status information to the client:

```
WM_DDE_ACK(wStatus, aMOUSE)
```

If Personal Communications cannot perform the DDE_UNADVISE, Personal Communications 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 | No | No |

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. |

| Return Code | Explanation |
|-------------|--------------------------|
| 9 | A system error occurred. |

Stop Session Advise

| | | |
|------|------|-----|
| 3270 | 5250 | VT |
| Yes | Yes | Yes |

The **Stop Session Advise** function disconnects a link between Personal Communications 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.

Personal Communications Response

If Personal Communications can perform the DDE_UNADVISE, Personal Communications returns an ACK message containing positive status information to the client:

```
WM_DDE_ACK(wStatus, aItem)
```

If Personal Communications cannot perform the DDE_UNADVISE, Personal Communications 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 Personal Communications 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 );
```

Personal Communications Response

Personal Communications acknowledges the terminate command with a terminate message:

WM_DDE_TERMINATE

Terminate Structured Field Conversation

| | | |
|------|------|----|
| 3270 | 5250 | VT |
| Yes | No | No |

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 );
```

Personal Communications Response

Personal Communications acknowledges the terminate command with this message:

WM_DDE_TERMINATE

When the user closes a Personal Communications session, any global memory blocks that were allocated by Personal Communications 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 | No | No |

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,
             PackDDELParm(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 Windows 32-Bit Environment

Personal Communications 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.

Personal Communications supports two kinds of DDE conversation. One is Personal Communications, which acts as a DDE menu client application, and the other is Personal Communications, which acts as a DDE menu server.

DDE Menu Client

To add, delete, and change menu items, the following DDE conversation must take place between the session and DDE menu server application.

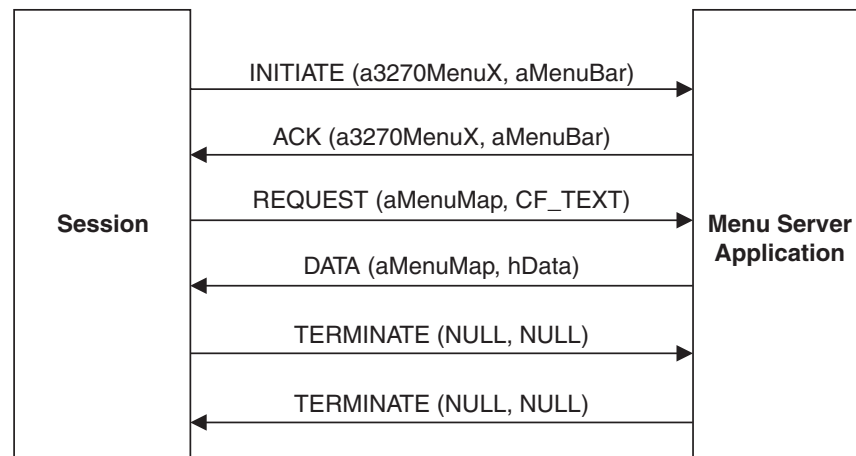


Figure 3. DDE Menu Server Conversation

The following data hierarchy details the menu map Personal Communications 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, Personal Communications 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, Personal Communications 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 Personal Communications the process is complete. Personal Communications will then reenable the window for the user.

DDE Menu Server

To add, delete, and change menu items, the following DDE conversation must take place between the session and a DDE menu client application.

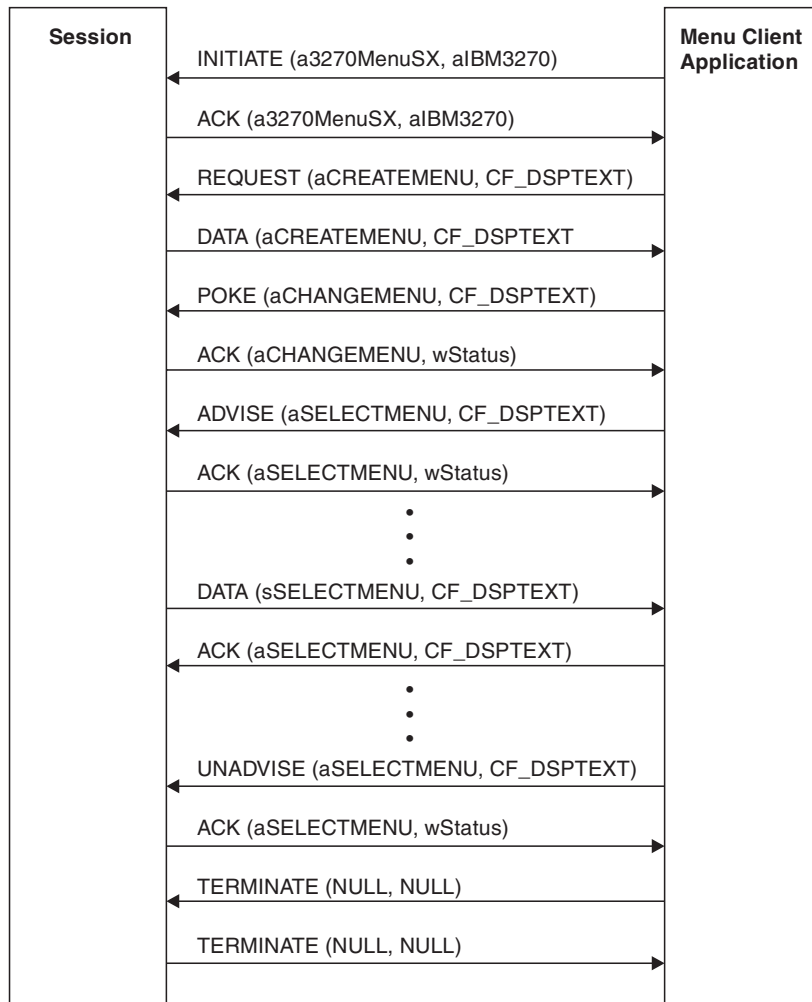


Figure 4. DDE Menu Client Conversation

When the user selects a menu item from the new menu, Personal Communications will send a DDE DATA with aSELECTMENU as the item. When Personal Communications sends DDE DATA to the client application, Personal Communications 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 Personal Communications the process is complete. Personal Communications will then reenable the window for the user.

DDE Menu Functions

The DDE Menu Item API functions listed below are available for use with Personal Communications. PC/3270 Windows mode and PC400 provide all of the following functions.

- “Change Menu Item” on page 275
- “Create Menu Item” on page 281
- “Initiate Menu Conversation” on page 282
- “Start Menu Advise” on page 282
- “Stop Menu Advise” on page 284
- “Terminate Menu Conversation” on page 284

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,
             PackDDEParam(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 check mark 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 check mark 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 Personal Communications 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 Personal Communications when **Create Menu Item** function is executed.

wFlags

The following options can be set:

| | |
|-----------------|--|
| MF_CHECKED | // Places a check mark 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 |

```

MF_UNCHECKED // ignored.
              // Does not place a check mark
              // 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 Personal Communications 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 Personal Communications 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 check mark 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

```

```

MF_MENUBREAK // from the old column with a
              // vertical line.
MF_SEPARATOR // 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_UNCHECKED // 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.
              // Does not place a check mark
              // 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 Personal Communications 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 Personal Communications 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 Personal Communications 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 check mark 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,
```

```

MF_UNCHECKED // or highlighted. The wIDNew
              // and szItemName fields are
              // ignored.
              // Does not place a check mark
              // 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 Personal Communications 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.

```

Personal Communications Response

Personal Communications receives the requests to change a menu and processes them. If the requests cannot be accepted, Personal Communications returns a negative ACK message containing one of the following status codes in the low-order byte of the wStatus word. Otherwise, Personal Communications 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 Personal Communications 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.

Personal Communications Response

Personal Communications returns the handle of the newly created menu item in a DDE data message if the Personal Communications 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             hMenuItem;    /* 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 Personal Communications 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 Personal Communications. 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.

Personal Communications Response

If Personal Communications can support a conversation with the client application, Personal Communications 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,
            PackDDE1Param(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.

Personal Communications Response

Personal Communications 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 Personal Communications.

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.

Personal Communications Response

If Personal Communications can perform the DDE_UNADVISE, Personal Communications returns an ACK message containing positive status information to the client:

```
WM_DDE_ACK(wStatus, aCLOSE)
```

If Personal Communications cannot perform the DDE_UNADVISE, Personal Communications 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 Personal Communications 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 );
```

Personal Communications Response

Personal Communications acknowledges the terminate command with this message:

WM_DDE_TERMINATE

Summary of DDE Functions in a Windows 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 20. DDE Function Summary

| Function Name | Client Command | Server Response |
|--|---|---|
| Code Conversion (system) | PostMessage(hServerWnd, WM_DDE_POKE, hClientWnd, PackDDEIParam(WM_DDE_POKE, hData, aCONV)); | UnPackDDEIParam(WM_DDE_ACK, wStatus, aCONV) |
| 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, aSYSICON)); | UnPackDDEIParam(WM_DDE_DATA, hData, aSYSICON) or UnPackDDEIParam(WM_DDE_ACK, wStatus, aSYSICON) |
| | cfFormat = CF_TEXT | |
| Get System Formats (system) | PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELPARAM(cfFormat, aFORMATS)); | UnPackDDEIParam(WM_DDE_DATA, hData, aFORMATS) or UnPackDDEIParam(WM_DDE_ACK, wStatus, aFORMATS) |
| | cfFormat = CF_TEXT | |
| Get System Status (system) | PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELPARAM(cfFormat, aSTATUS)); | UnPackDDEIParam(WM_DDE_DATA, hData, aSTATUS) or UnPackDDEIParam(WM_DDE_ACK, wStatus, aSTATUS) |
| | cfFormat = CF_TEXT | |
| Get System SysItems (system) | PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELPARAM(cfFormat, aSYSITEMS)); | UnPackDDEIParam(WM_DDE_DATA, hData, aSYSITEMS) or UnPackDDEIParam(WM_DDE_ACK, wStatus, aSYSITEMS) |
| | cfFormat = CF_TEXT | |
| Get System Topics (system) | PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELPARAM(cfFormat, aTOPICS)); | UnPackDDEIParam(WM_DDE_DATA, hData, aTOPICS) or UnPackDDEIParam(WM_DDE_ACK, wStatus, aTOPICS) |
| | cfFormat = CF_TEXT | |
| Terminate System Conversation (system) | SendMessage(hServerWnd, WM_DDE_TERMINATE, hClientWnd, 0); | WM_DDE_TERMINATE |

Table 20. DDE Function Summary (continued)

| Function Name | Client Command | Server Response |
|--|--|---|
| Initiate Session Conversation (session) | SendMessage(-1, WM_DDE_INITIATE, hClientWnd, MAKELPARAM(aIBM327032,aSessionN)); | LOWORD/HIWORD to unpack WM_DDE_ACK(aIBM327032,aSessionN) |
| | N = a session letter A through Z. | |
| Find Field (session) | PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELPARAM(cfFormat,aFIELD)); | UnPackDDElParam(WM_DDE_DATA,hData,aFIELD) or UnPackDDElParam(WM_DDE_ACK,wStatus,aFIELD) |
| | cfFormat = CF_DSPTTEXT | |
| Get Keystrokes (session) | PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELPARAM(cfFormat,aKEYS)); | UnPackDDElParam(WM_DDE_DATA,hData,aKEYS) or UnPackDDElParam(WM_DDE_ACK,wStatus,aKEYS) |
| | cfFormat = CF_DSPTTEXT | |
| Get Mouse Input (session) | PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELPARAM(cfFormat,aMOUSE)); | UnPackDDElParam(WM_DDE_DATA,hData,aMOUSE) or UnPackDDElParam(WM_DDE_ACK,wStatus,aMOUSE) |
| | cfFormat = CF_TEXT CF_DSPTTEXT | |
| Get Number of Close Requests (session) | PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELPARAM(cfFormat,aCLOSE)); | UnPackDDElParam(WM_DDE_DATA,hData,aCLOSE) or UnPackDDElParam(WM_DDE_ACK,wStatus,aCLOSE) |
| | cfFormat = CF_DSPTTEXT | |
| Get Operator Information Area (session) | PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELPARAM(cfFormat,aOIA)); | UnPackDDElParam(WM_DDE_DATA,hData,aOIA) or UnPackDDElParam(WM_DDE_ACK,wStatus,aOIA) |
| | cfFormat = CF_DSPTTEXT | |
| Get Partial Presentation Space (session) | PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELPARAM(cfFormat,aEPS)); | UnPackDDElParam(WM_DDE_DATA,hData,aEPS) or UnPackDDElParam(WM_DDE_ACK,wStatus,aEPS) |
| | cfFormat = CF_TEXT CF_DSPTTEXT | |
| Get Presentation Space (session) | PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELPARAM(cfFormat,aPS)); | UnPackDDElParam(WM_DDE_DATA,hData,aPS) or UnPackDDElParam(WM_DDE_ACK,wStatus,aPS) |
| | cfFormat = CF_TEXT CF_DSPTTEXT | |
| Get Session Status (session) | PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELPARAM(cfFormat,aSSTAT)); | UnPackDDElParam(WM_DDE_DATA,hData,aSSTAT) or UnPackDDElParam(WM_DDE_ACK,wStatus,aSSTAT) |
| | cfFormat = CF_TEXT | |
| Get Trim Rectangle (session) | PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELPARAM(cfFormat,aTRIMRECT)); | UnPackDDElParam(WM_DDE_DATA,hData,aTRIMRECT) or UnPackDDElParam(WM_DDE_ACK,wStatus,aTRIMRECT) |
| | cfFormat = CF_TEXT | |

Table 20. DDE Function Summary (continued)

| Function Name | Client Command | Server Response |
|--|--|--|
| Put Data to Presentation Space (session) | PostMessage(hServerWnd, WM_DDE_POKE, hClientWnd, PackDDE1Param(WM_DDE_POKE, hData, aEPS)); | UnPackDDE1Param(WM_DDE_ACK, wStatus, aEPS) |
| | hData = Handle to a global memory object | |
| Search for String (session) | PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELPARAM(cfFormat, aSTRING)); | UnPackDDE1Param(WM_DDE_DATA, hData, aSTRING) or UnPackDDE1Param(WM_DDE_ACK, wStatus, aSTRING) |
| | cfFormat = CF_DSPTXT | |
| Send Keystrokes (session) | PostMessage(hServerWnd, WM_DDE_POKE, hClientWnd, PackDDE1Param(WM_DDE_POKE, hData, aKEYS)); | UnPackDDE1Param(WM_DDE_ACK, wStatus, aKEYS) |
| | hData = Handle to a global memory object | |
| Session Execute Macro (session) | PostMessage(hServerWnd, WM_DDE_EXECUTE, hClientWnd, (LPARAM)hCommands); | UnPackDDE1Param(WM_DDE_ACK, wStatus, NULL) |
| | hCommands = Handle to a global memory object | |
| Set Cursor Position (session) | PostMessage(hServerWnd, WM_DDE_POKE, hClientWnd, PackDDE1Param(WM_DDE_POKE, hData, aSETCURSOR)); | UnPackDDE1Param(WM_DDE_ACK, wStatus, aSETCURSOR) |
| | hData = Handle to a global memory object | |
| Set Mouse Intercept Condition (session) | PostMessage(hServerWnd, WM_DDE_POKE, hClientWnd, PackDDE1Param(WM_DDE_POKE, hData, aMOUSE)); | UnPackDDE1Param(WM_DDE_ACK, wStatus, aMOUSE) |
| | cfFormat = CF_TEXT CF_DSPTXT hData = Handle to a global memory object | |
| Set Presentation Space Service Condition (session) | PostMessage(hServerWnd, WM_DDE_POKE, hClientWnd, PackDDE1Param(WM_DDE_POKE, hData, aEPSCOND)); | UnPackDDE1Param(WM_DDE_ACK, wStatus, aEPSCOND) |
| | hData = Handle to a global memory object | |
| Set Session Advise Condition (session) | PostMessage(hServerWnd, WM_DDE_POKE, hClientWnd, PackDDE1Param(WM_DDE_POKE, hData, aPSCOND)); | UnPackDDE1Param(WM_DDE_ACK, wStatus, aPSCOND) |
| | hData = Handle to a global memory object | |

Table 20. DDE Function Summary (continued)

| Function Name | Client Command | Server Response |
|--|---|---|
| Start Close Intercept (session) | SendMessage(hServerWnd, WM_DDE_ADVISE, hClientWnd, PackDDE1Param(WM_DDE_ADVISE, hOptions, aCLOSE)); | UnPackDDE1Param(WM_DDE_ACK, wStatus, aCLOSE) or UnPackDDE1Param(WM_DDE_DATA, hData, aCLOSE) |
| | hOptions = Handle to a global memory object | |
| Start Keystroke Intercept (session) | SendMessage(hServerWnd, WM_DDE_ADVISE, hClientWnd, PackDDE1Param(WM_DDE_ADVISE, hOptions, aKEYS)); | UnPackDDE1Param(WM_DDE_ACK, wStatus, aKEYS) or UnPackDDE1Param(WM_DDE_DATA, hData, aKEYS) |
| | hOptions = Handle to a global memory object | |
| Start Mouse Input Intercept (session) | SendMessage(hServerWnd, WM_DDE_ADVISE, hClientWnd, PackDDE1Param(WM_DDE_ADVISE, hOptions, aMOUSE)); | UnPackDDE1Param(WM_DDE_ACK, wStatus, aMOUSE) or UnPackDDE1Param(WM_DDE_DATA, hData, aMOUSE) |
| | hOptions = Handle to a global memory object | |
| Start Session Advise (session) | PostMessage(hServerWnd, WM_DDE_ADVISE, hClientWnd, PackDDE1Param(WM_DDE_ADVISE, hOptions, aItem)); | UnPackDDE1Param(WM_DDE_ACK, wStatus, aItem) or UnPackDDE1Param(WM_DDE_DATA, hData, aItem) |
| | hOptions = Handle to a global memory object aItem = OIA PS TRIMRECT | |
| Stop Close Intercept (session) | PostMessage(hServerWnd, WM_DDE_UNADVISE, hClientWnd, MAKELPARAM(NULL, aCLOSE)); | UnPackDDE1Param(WM_DDE_ACK, wStatus, aCLOSE) |
| Stop Keystroke Intercept (session) | PostMessage(hServerWnd, WM_DDE_UNADVISE, hClientWnd, MAKELPARAM(NULL, aKEYS)); | UnPackDDE1Param(WM_DDE_ACK, wStatus, aKEYS) |
| Stop Mouse Input Intercept (session) | PostMessage(hServerWnd, WM_DDE_UNADVISE, hClientWnd, MAKELPARAM(NULL, aMOUSE)); | UnPackDDE1Param(WM_DDE_ACK, wStatus, aMOUSE) |
| Stop Session Advise (session) | PostMessage(hServerWnd, WM_DDE_UNADVISE, hClientWnd, MAKELPARAM(NULL, aItem)); | UnPackDDE1Param(WM_DDE_ACK, wStatus, aItem) |
| | aItem = SysItems Topics NULL | |
| Terminate Session Conversation (session) | SendMessage(hServerWnd, WM_DDE_TERMINATE, hClientWnd, 0); | WM_DDE_TERMINATE |

Table 20. DDE Function Summary (continued)

| Function Name | Client Command | Server Response |
|--|---|---|
| Initiate Structured Field Conversation (structured field) | SendMessage(-1, WM_DDE_INITIATE, hClientWnd, MAKELPARAM(aIBM327032, aLUN_xxxx)); | LOWORD/HIWORD to unpack WM_DDE_ACK(aIBM327032, aLUN_xxxx) |
| | N = a session letter A through Z. xxxx = a user defined string. | |
| 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, PackDDE1Param(WM_DDE_POKE, hData, aSFCOND)); | UnPackDDE1Param(WM_DDE_ACK, wStatus, aSFCOND) |
| | hData = Handle to a global memory object | |
| Start Read SF (structured field) | PostMessage(hServerWnd, WM_DDE_ADVISE, hClientWnd, PackDDE1Param(WM_DDE_ADVISE, hOptions, aSF)); | UnPackDDE1Param(WM_DDE_ACK, wStatus, aSF) or UnPackDDE1Param(WM_DDE_DATA, hData, aSF) |
| | hOptions = Handle to a global memory object | |
| Stop Read SF (structured field) | PostMessage(hServerWnd, WM_DDE_UNADVISE, hClientWnd, MAKELPARAM(NULL, aSF)); | UnPackDDE1Param(WM_DDE_ACK, wStatus, aSF) |
| Write SF (structured field) | PostMessage(hServerWnd, WM_DDE_POKE, hClientWnd, PackDDE1Param(WM_DDE_POKE, hData, aSF)); | UnPackDDE1Param(WM_DDE_ACK, wStatus, aSF) |
| | hData = Handle to a global memory object | |
| Initiate Menu Conversation (menu) | SendMessage(-1, WM_DDE_INITIATE, hClientWnd, MAKELPARAM(aIBM327032, a3270MenuSN)); | LOWORD/HIWORD to unpack WM_DDE_ACK(aIBM327032, a3270MenuSN) |
| | N = a session letter A through Z | |
| Change Menu Item (menu) | PostMessage(hServerWnd, WM_DDE_POKE, hClientWnd, PackDDE1Param(WM_DDE_POKE, hData, aCHANGEMENU)); | UnPackDDE1Param(WM_DDE_ACK, wStatus, aCHANGEMENU) |
| | hData = Handle to a global memory object | |
| Create Menu Item (menu) | PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELPARAM(cfFormat, aCREATEMENU)); | UnPackDDE1Param(WM_DDE_DATA, hData, aCREATEMENU) or UnPackDDE1Param(WM_DDE_ACK, wStatus, aCREATEMENU) |
| | cfFormat = CF_DSPTEXT | |

Table 20. DDE Function Summary (continued)

| Function Name | Client Command | Server Response |
|------------------------------------|---|---|
| Start Menu Advise (menu) | <pre>PostMessage(hServerWnd, WM_DDE_ADVISE, hClientWnd, PackDDElParam(WM_DDE_ADVISE, hOption,aSELECTMENU));</pre> | UnPackDDElParam(WM_DDE_ACK,wStatus,aSELECTMENU) or UnPackDDElParam(WM_DDE_DATA,hData,aSELECTMENU) |
| | <pre>hData = Handle to a global memory object</pre> | |
| Stop Menu Advise (menu) | <pre>PostMessage(hServerWnd, WM_DDE_UNADVISE, hClientWnd, MAKELPARAM(NULL,aSELECTMENU));</pre> | UnPackDDElParam(WM_DDE_ACK,wStatus,aCLOSE) |
| Terminate Menu Conversation (menu) | <pre>SendMessage(hServerWnd, WM_DDE_TERMINATE, hClientWnd, 0);</pre> | WM_DDE_TERMINATE |

Chapter 7. Using DDE Functions with a DDE Client Application

Windows allows users to run multiple 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.

Personal Communications 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 Personal Communications DDE Interface

To start a DDE conversation and data exchange with Personal Communications, client application programs need to know the application program name, topic name, and item name that Personal Communications can recognize. The exchange data type between an application program and Personal Communications is defined with the combination of these names.

Table 21. Naming Scheme for Data Items

| Level | Description | Example |
|-------------|---|-------------------------|
| Application | A Windows task or a specific task of the application. In this book, application programs are Personal Communications. | 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, Personal Communications 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 22. Topics for Personal Communications

| Topic | Conversation Name | Conversation Type |
|-----------------------------------|----------------------|------------------------|
| System | System conversation | Cold link |
| SessionA, SessionB, ..., SessionZ | Session conversation | Cold link and hot link |

Table 22. Topics for Personal Communications (continued)

| 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 Personal Communications. 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 Personal Communications 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 Personal Communications

To use the DDE interface with Personal Communications, the client application should first start a DDE conversation with Personal Communications. To start a DDE conversation, specify *IBM327032* or *IBM525032* as an application name for 32-bit applications; *IBM3270* or *IBM5250* 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 Personal Communications

To complete the conversation, the client application needs to terminate the DDE conversation with Personal Communications. To terminate the conversation, use the DDE function (Terminate) in the client application.

Session Conversation

To use the Personal Communications 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 Personal Communications sessions, the client application should start the DDE conversation with Personal Communications. 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 Personal Communications 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 Personal Communications session window using the DDE functions. Specify the command in the DDE function (Execute) of the client application. See "Session Execute Macro" on page 244 for details.

Terminating the DDE Session Conversation

The client application should terminate the DDE conversation with Personal Communications when completing the task. To terminate the conversation, use the DDE function (Terminate) in the client application.

Session Conversation (Hot Link)

To use the Personal Communications 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 Personal Communications sessions, the client application should start the DDE conversation with Personal Communications. 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 Personal Communications when completing the task. To terminate the conversation, use the DDE function (Terminate) in the client application.

Personal Communications 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

- "Initiate System Conversation" on page 297
- "Get System Configuration"
- "Get System Formats" on page 296
- "Get System Status" on page 296
- "Get System SysItems" on page 297
- "Get System Topics" on page 297
- "Terminate System Conversation" on page 298

DDE functions for session conversation

- "Initiate Session Conversation" on page 304 *1
- "Find Field" on page 298
- "Get Operator Information Area" on page 299
- "Get Partial Presentation Space" on page 300
- "Get Presentation Space" on page 301
- "Get Session Status" on page 302
- "Get Trim Rectangle" on page 302
- "Put Data to Presentation Space" on page 305
- "Search for String" on page 305
- "Session Execute Macro" on page 306
- "Set Cursor Position" on page 307
- "Terminate Session Conversation" on page 307 *2

DDE functions for session conversation (hot link)

- "Initiate Session Conversation" on page 307 (same as *1)
- "Start Close Intercept" on page 308
- "Start Keystroke Intercept" on page 308
- "Start Session Advise" on page 309
- "Stop Close Intercept" on page 310
- "Stop Keystroke Intercept" on page 311
- "Stop Session Advise" on page 311
- "Terminate Session Conversation" on page 311 (same as *2)

DDE Functions for System Conversation

The following DDE functions are provided for Personal Communications system conversation.

Get System Configuration

The **Get System Configuration** function returns the Personal Communications support level and other system-related values.

| DDE Parameter | Value |
|---------------|--------|
| Item | SysCon |

The client application can use the Personal Communications DDE function by specifying the DDE function (Request) of the client application with the topic name (System).

Personal Communications Response

The Personal Communications system returns the Personal Communications system configuration data item.

Returned Information: See “Get System Configuration” on page 234 for details.

If Personal Communications 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 Personal Communications.

| DDE Parameter | Value |
|---------------|---------|
| Item | Formats |

The client application can use the Personal Communications DDE function by specifying the DDE function (Request) of the client application with the item name (Formats).

Personal Communications Response

Personal Communications returns a list of supported Windows Clipboard formats.

If Personal Communications 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 Personal Communicationssession.

| DDE Parameter | Value |
|---------------|--------|
| Item | SysCon |

The client application can use the Personal Communications DDE function by specifying the DDE function (Request) of the client application with the item name (SysCon).

Personal Communications Response

Personal Communications returns a series of status information to each open session.

Returned Information: See “Get System Status” on page 236 for details.

If Personal Communications 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 Personal Communications system topic.

| DDE Parameter | Value |
|---------------|----------|
| Item | SysItems |

The client application can use the Personal Communications DDE function by specifying the DDE function (Request) of the client application with the item name (SysItems).

Personal Communications Response

Personal Communications returns a list of Personal Communications system topic data items. The following data items are supported by Personal Communications:

- SysItems
- Topics
- Status
- Formats
- SysCon

If Personal Communications 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 Personal Communications.

| DDE Parameter | Value |
|---------------|--------|
| Item | Topics |

The client application can use the Personal Communications DDE function by specifying the DDE function (Request) of the client application with the item name (Topics).

Personal Communications Response

The following topics are supported by Personal Communications:

- System
- SessionA, SessionB, ..., SessionZ

If Personal Communications 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 Personal Communications 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 Personal Communications 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 "Tb" "Pb" "Nb" "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 Personal Communications DDE function by specifying the DDE function (Request) of the client application with the foregoing item name.

Personal Communications 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. |
| Unprotected/Protected | N | 0 = Unprotected data field. 1 = Protected data field. |
| A/N | N | 0 = Alphanumeric. 1 = Numeric. |

| Field | Returned Information | Explanation |
|--------------------|----------------------|---|
| 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 = Signed numeric: numbers 0–9, plus, minus, and Dup key allowed. |
| MDT | N | 0 = Field is not changed. 1 = Field is changed. |
| Field start offset | NNNN | Field starts this field position. |
| Field length | NNNN | Field length. |

If Personal Communications 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 Personal Communications DDE function by specifying the DDE function (Request) of the client application with the item name (OIA).

Personal Communications Response

The following table shows the OIA information that Personal Communications 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 Personal Communications 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 |

| Parameter | Value | Explanation |
|-----------|--------|-----------------------------|
| 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 Personal Communications DDE function by specifying the DDE function (Request) of the client application with the foregoing item names.

Personal Communications Response

The following table shows the information Personal Communications 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 Personal Communications 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 Personal Communications DDE function by specifying the DDE function (Request) of the client application with the item name (PS).

Personal Communications Response

The following table shows the information Personal Communications 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 Personal Communications 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 Personal Communications DDE function by specifying the DDE function (Request) of the client application with the item name (SSTAT).

Personal Communications Response

Refer to “Get Session Status” on page 371 for the returned information.

If Personal Communications 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 Personal Communications DDE function by specifying the DDE function (Request) of the client application with the item name (TRIMRECT).

Trim Rectangle Word by Word

This feature enables you to resize the trim rectangle along word boundaries. The Mark Word Left function moves the left side of the rectangle to the front of the previous word in the first line of the rectangle. The Mark Word Right function moves the right side of the rectangle to the front of the next word in the first line of the rectangle.

These functions are available only for single-byte languages. They are not available for Thai, Hindi, double-byte languages, or bidirectional sessions.

You can enable the Trim Rectangle Word by Word functions by adding the following keyword to the PCSWIN.INI file:

```
[Edit]
TrimRectJumpToWord=Y
```

When the feature is enabled, Mark Word Left can be invoked with the key combination Ctrl+Shift+LeftArrow. Mark Word Right can be invoked with the Ctrl+Shift+Right Arrow key combination.

The Mark Word Left and Mark Word Right functions can also be added to the popup keypad, a macro or script, or to a mouse button.

Table 23. Mark Word Left

| | |
|-------------------------|------------------|
| Token name | mark word left |
| Face name | MrkWd< |
| | |
| Context | Available |
| Keyboard | Yes |
| Popup keypad | Yes |
| Mouse | Yes |
| Macro | Yes |
| Keyboard/macro function | Yes |
| 3270 SBCS | Yes |
| 3270 DBCS | No |
| 3270 AEA (SBCS) | Yes |
| 5250 SBCS | Yes |
| 5250 DBCS | No |
| VT SBCS | Yes |

Table 24. Mark Word Right

| | |
|----------------|------------------|
| Token name | mark word right |
| Face name | MrkWd> |
| | |
| Context | Available |
| Keyboard | Yes |

Table 24. Mark Word Right (continued)

| | |
|-------------------------|-----|
| Popup keypad | Yes |
| Mouse | Yes |
| Macro | Yes |
| Keyboard/macro function | Yes |
| 3270 SBCS | Yes |
| 3270 DBCS | No |
| 3270 AEA (SBCS) | Yes |
| 5250 SBCS | Yes |
| 5250 DBCS | No |
| VT SBCS | Yes |

Autocopy

This feature enables you to automatically copy the trim rectangle text to the clipboard.

After you mark an area on the screen with a trim rectangle, the text inside the Trim Rectangle is automatically copied to the clipboard. When an existing trim rectangle is moved to another screen area, the text inside the new trim rectangle is automatically copied to the clipboard.

Enable the feature by adding the following keywords to the .WS profile. Both keywords must be enabled in order for Autocopy to work correctly.

```
[Edit]
AutoCopy=Y
TrimRectRemainAfterEdit=Y
```

Personal Communications Response

The information returned from Personal Communications is as follows:

| Field | Returned Information | Explanation |
|-------|----------------------|------------------------------------|
| PS | | PS data starts from this position. |

If Personal Communications 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).

Personal Communications Response

If a topic is not specified, Personal Communications 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 <table> <tr> <td>1</td> <td>Yes</td> </tr> <tr> <td>0</td> <td>No</td> </tr> </table> | 1 | Yes | 0 | No |
| 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 Personal Communications DDE function by specifying the DDE function (Poke) of the client application with the foregoing item name.

Personal Communications Response

If Personal Communications 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 <ul style="list-style-type: none"> • 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: <i>This is a double quotation" mark.</i> is specified as <i>"This is a double quotation"" mark."</i> |

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 Personal Communications DDE function by specifying the DDE function (Request) of the client application with the foregoing item names.

Personal Communications Response

Personal Communications returns the following information:

| Field | Returned Information | Explanation |
|---------------------|----------------------|--|
| String start offset | NNNN, None | "None" is returned if the string is not found. |

If Personal Communications 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 Personal Communications.

Refer to "Session Execute Macro" on page 382 for details of commands and macro strings.

The client application can use the Personal Communications DDE function by specifying the DDE function (Execute) of the client application.

Personal Communications Response

A system error can cause Personal Communications 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 Personal Communications DDE function by specifying the DDE function (Poke) of the client application with the foregoing item names.

Personal Communications Response

If Personal Communications 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 Personal Communications.

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 Personal Communications 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 Personal Communications 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 Personal Communications DDE function by specifying the DDE function (Advise) of the client application with the foregoing item name.

Personal Communications Response

Personal Communications 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 Personal Communications 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 Personal Communications DDE function by specifying the DDE function (Advise) of the client application with the foregoing item name.

Personal Communications Response

Personal Communications returns the following information:

| Field | Returned Information | Explanation |
|-------|----------------------|--------------------------------|
| Keys | | Refer to Table 19 on page 247. |

If Personal Communications 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 Personal Communications. 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 <ul style="list-style-type: none"> • 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: <i>This is a double quotation" mark.</i> is specified as <i>"This is a double quotation"" mark."</i> |

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 Personal Communications DDE function by specifying the DDE function (Advise) of the client application with the foregoing item name.

Personal Communications Response

Refer to “Get Partial Presentation Space” on page 300, “Get Operator Information Area” on page 299, and “Get Trim Rectangle” on page 302.

If Personal Communications 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 Personal Communications DDE function by specifying the DDE function, **Unadvise**, of the client application with the foregoing item name.

Personal Communications Response

If Personal Communications 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 Keystroke Intercept** function, the client application stops intercepting the keystrokes.

| DDE Parameter | Value |
|---------------|-------|
| Item | KEYS |

The client application can use the Personal Communications DDE function by specifying the DDE function, **Unadvise**, of the client application with the foregoing item name.

Personal Communications Response

If Personal Communications 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 Personal Communications.

| 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 Personal Communications DDE function by specifying the DDE function, **Unadvise**, of the client application with the foregoing item name.

Personal Communications Response

If Personal Communications 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 Personal Communications 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 Personal Communications'   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 Personal Communications'   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 Personal Communications'   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
'   Personal Communications 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

ErrorHandler:
    FunctionComp& = False
    Resume Next
End Sub

'*****
'***                                     ***
'***   Get System Topics                 ***
'***                                     ***
'*****
'
'   Requests a list of Personal Communications'   Topics
'
Sub Command2_Click ()
On Error GoTo ErrorHandler
    FunctionComp& = True

    Text1.LinkItem = "Topics"
    Text1.LinkRequest

    If FunctionComp&= False Then
        MsgBox "Error has occurred", 48, "DDE sample"
    End If
Exit Sub

ErrorHandler:
    FunctionComp& = False
    Resume Next
End Sub

'*****
'***                                     ***
'***   Terminate System Conversation     ***
'***                                     ***
'*****
'
'   Terminates DDE Conversation with system
'
Sub Command3_Click ()
On Error GoTo ErrorHandler
    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

ErrorHandlerTerm:
    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

ErrorHandler:
    FunctionComp& = False
    Resume Next
End Sub

'*****
'***                                     ***
'***   Search for String                 ***
'***                                     ***
'*****

'
'   Search forward for string "Hello!" from PS Position 1
'
Sub Command2_Click ()
On Error GoTo ErrorHandler
    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

ErrorHandler:
    FunctionComp& = False
    Resume Next
End Sub

'*****
'***                                     ***
'***   Session Execute Macro           ***
'***                                     ***
'*****

'
'   Maximize the Session
'
Sub Command2_Click ()
On Error GoTo ErrorHandler
    FunctionComp& = True

    Text1.LinkExecute "[WINDOW(MAXIMIZE)]"

    If FunctionComp&= False Then
        MsgBox "Error has occurred", 48, "DDE sample"
    End If
    Exit Sub

ErrorHandler:
    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      = "RIC1"
    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 8. 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.

Note: SRPI is not available on Personal Communications for iSeries and will not work when connected to an iSeries, eServer i5, or System i5 host.

PC/3270 SRPI supports 32-bit SRPI Requester Program written in C or C++.

How to Use SRPI

You can write the application program using the SRPI in C or C++. To develop a SRPI application, do as follows:

1. Prepare the source code and add the appropriate SRPI calls.
2. Include the header file `UCCPRB.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 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.

`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.

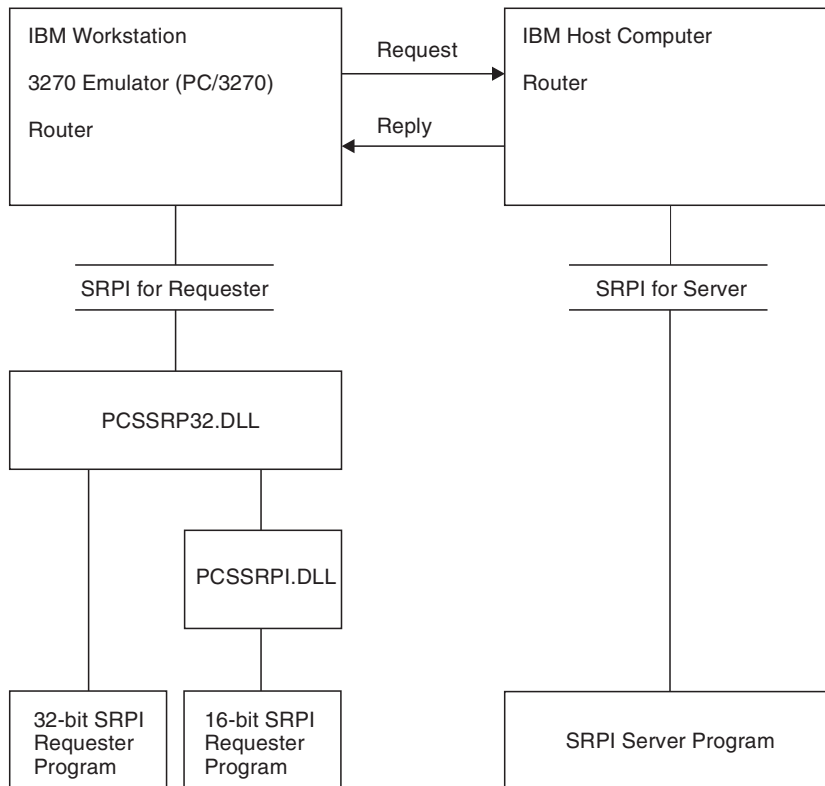


Figure 5. Example of PC/3270 SRPI Requester and Server

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.

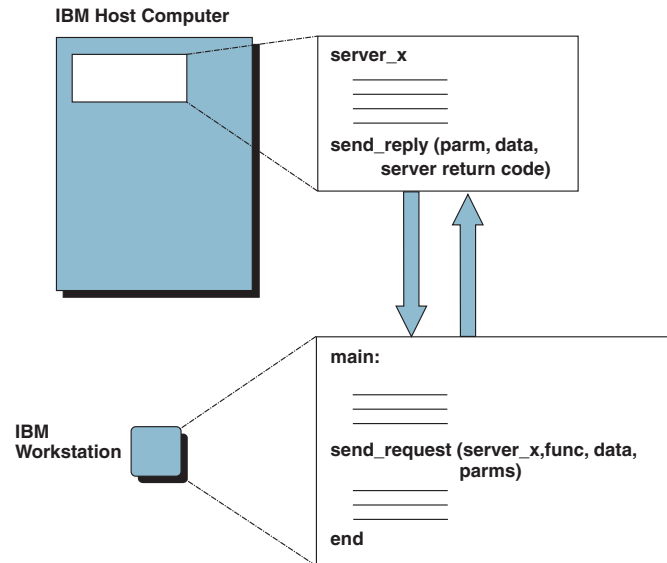


Figure 6. IBM Workstation Requester and IBM Host Computer Server Relationship

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 an illustration of the requester and server flow.

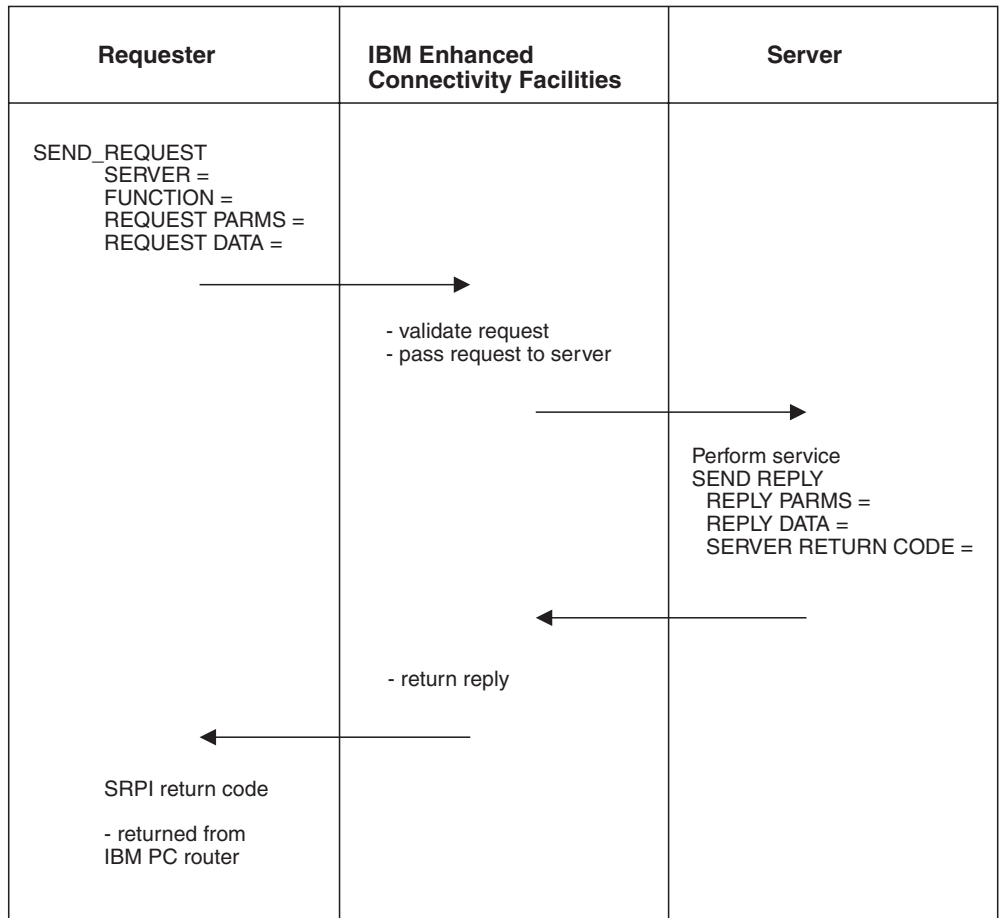


Figure 7. Example of an SRPI Requester and Server Flow

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 25 on page 326 and in Table 26 on page 328.

Supplied Parameters

Table 25. Parameters Supplied by the SRPI Requester

| Name of Parameter | Required/Optional | Default Value | Description |
|-------------------|-------------------|---------------|---|
| 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. |

Table 25. Parameters Supplied by the SRPI Requester (continued)

| 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. |
| 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. |
| 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. |
| 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. |

Table 25. Parameters Supplied by the SRPI Requester (continued)

| 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. |

Returned Parameters

Table 26. 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 D, 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 the Personal Communications installation.

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 **UCCPRB.H** file. For the definitions and value ranges of the supplied and returned parameters, see "Supplied Parameters" on page 326 and "Returned Parameters" on page 328.

SRPI Return Codes

See Appendix D, "SRPI Return Codes," on page 349 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. Refer to *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 27. 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 |

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 28. 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 29. 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.

Refer to *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 30. 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.

Refer to *IBM 3270 Information Display System Data Stream Programmer's Reference* for the field definitions for this query reply.

Table 31. 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 |

Table 31. Base DDM Query Reply Format with Direct Access and Name Self-Defining Parameters (continued)

| Offset | Length | Content | Meaning |
|-------------|---------|--------------|--|
| 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.

Refer to *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, refer to *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 32. 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 | FLAGS | Reserved |
| | BIT 0 | QUERY B'1' | Read Part (Query, Query List) Auxiliary device supports Query |
| | 1-7 | RES | Reserved, must be B'0's |
| 5 | 1 byte | FLAGS | Reserved |
| 6-7 | 2 bytes | LIMIN | Maximum DDM bytes allowed in inbound transmission |
| 8-9 | 2 bytes | LIMOUT | Maximum DDM bytes allowed in outbound transmission |
| 10 | 1 byte | TYPE X'01' X'02' Others | Type of auxiliary device supported IBM auxiliary device display IBM auxiliary device printer Reserved |
| 11 | 1 byte | X'04' | Parameter length |
| 12 | 1 byte | X'01' | Direct access |
| 13-14 | 1 word | DOID | Destination/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 33 on page 336.

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 33. 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 34. 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.

Refer to *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 35. 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 |

Table 35. OEM Auxiliary Device Base Format with Direct Access Self-Defining Parameter (continued)

| Offset | Length | Content | Meaning |
|--------|--------|---------|---|
| 24–25 | 1 word | DOID | Destination/origin ID assigned by the subsystem |

Refer to *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 36.

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 36. 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 37. 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 38. 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 |

Refer to *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 39. 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 40. 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 <i>IBM 5080 Graphics System Principles of Operation</i> |
| | 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 <i>IBM RT PC Workstation Host Interface Program Version 1.1 User's Guide and Reference Manual</i> |
| | 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 41 on page 340.

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 41. 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 42. 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 43 on page 341.

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 43. 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.

Refer to *IBM 3270 Information Display System Data Stream Programmer's Reference* for the field definitions for this query reply.

Appendix B. Differences from Communication Manager/2 EHELLAPI

This appendix describes the differences between EHELLAPI of Personal Communications and EHELLAPI for Communication Manager/2.

The following EHELLAPI functions are different from those with the same names in Communication Manager/2. You need to understand the differences when you use these functions:

- **Set Session Parameter** (9)
- **Copy OIA** (13)
- **Copy String to PS** (15)
- **Storage Manager** (17)
- **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 Communication Manager:

OLDOIA, NEWOIA
COMPCASE, COMPICASE
OLD5250OIA, NEW5250OIA

Return Parameters

When the **Set Session Parameter** (9) function is terminated, Communication 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 Communication Manager/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.

Copy OIA (13)

The **Copy OIA (13)** function has the following differences between Communication Manager/2 and Personal Communications. For more information of the group and the column positions, refer to “Copy OIA (13)” on page 48.

- Byte Position 21
 - Personal Communications returns X'F6'.
 - Communication Manager/2 returns X'20'.
- Byte Positions 61–63
 - Personal Communications does not return the printer information.
 - Communication Manager/2 returns the printer information.
- Group 3: Shift State

Communication Manager/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).
 - Communication Manager/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).
 - Communication Manager/2 can return bits 1 and 2.
- Group 8 Byte 4: Input Inhibited
 - Personal Communications does not return bit 2 (System wait).
 - Communication Manager/2 can return bit 2.
- Group 10: Highlight Group 2
 - Personal Communications does not return bit 0 (Selected).
 - Communication Manager/2 can return bit 0.
- Group 11: Color Group 2
 - Personal Communications does not return bit 0 (Selected).
 - Communication Manager/2 can return bit 0.
- Group 13: Printer Status
 - In Personal Communications, this group is reserved.
 - Communication Manager/2 can return this group.
- Group 14: Graphics

Communication Manager/2 does not return bit 0 (Graphic cursor).

Copy String to PS (15)

In Communication Manager/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 Communication Manager/2, use the PUTEAB option of **Set Session Parameter (9)**.

Storage Manager (17)

Storage Manager (17) function provided by Communication Manager/2 is not supported by Personal Communications. Use the APIs provided by Windows to allocate the memory for the applications.

Copy String to Field (33)

In Communication Manager/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 Communication Manager/2, use the PUTEAB option of **Set Session Parameter (9)**.

Get Key (51)

Communication Manager/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)

EHLAPI 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 Communication Manager/2 implementation. However, Personal Communications for Windows EHLAPI returns the handle of the frame window. EHLAPI applications written for Personal Communications for Windows using this function need to use the parent of the window handle returned.

Query Sessions (10)

In Communication Manager/2, the descriptor for personal computer is returned. However, the descriptor is not returned in Personal Communications.

Connect for Structured Fields (120)

The event object for communication connection status provided by Communication Manager/2 is not in Personal Communications.

Allocate Communications Buffer (123)

In Communication Manager/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) |

Get Request Completion (125)

Personal Communications does not support a blank or null session ID.

Appendix C. DOS-Mode EHLLAPI for Windows

Personal Communications supports EHLLAPI applications for DOS. This appendix provides information about this support.

Installation

To install DOS EHLLAPI support for Personal Communications 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.

This procedure adds the following statement in config.nt.

```
device=%SystemRoot%\system32\drivers\h1ldrivr.com
```

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.

Appendix D. 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.

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 44.

Table 44. 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. |

Table 44. Type 1 Return Code Definitions and Descriptions (continued)

| Hexadecimal Return Code | Constant Return Code | Description |
|-------------------------|----------------------|---|
| X'0100 0604' | UERERRT1RPLEN | Reply parameters buffer length exceeds the maximum value. The maximum value allowed is 32763. |
| 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: <ul style="list-style-type: none"> • 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: <ul style="list-style-type: none"> • 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: <ul style="list-style-type: none"> • 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: <ul style="list-style-type: none"> • 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. |

Table 44. Type 1 Return Code Definitions and Descriptions (continued)

| Hexadecimal Return Code | Constant Return Code | Description |
|-------------------------|----------------------|---|
| 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. |
| 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: <ul style="list-style-type: none"> • 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 45.

Table 45. 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 46.

Table 46. Class Definitions for Type 2 and Type 3

| Value | Definition |
|----------------|------------|
| X'00' to X'01' | Reserved |
| X'02' | Syntax |

Table 46. Class Definitions for Type 2 and Type 3 (continued)

| Value | Definition |
|----------------|------------|
| 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 47.

Table 47. 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 48.

Table 48. Exception Object Values for Type 2 and Type 3

| Value | Definition |
|-------|---------------|
| X'00' | Not specified |
| X'01' | Prefix |

Table 48. Exception Object Values for Type 2 and Type 3 (continued)

| Value | Definition |
|-------|--------------------|
| 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 E. 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. Personal Communications is a DDE server for the open sessions that Personal Communications is 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 7, "Using DDE Functions with a DDE Client Application," on page 291.

Personal Communications 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 49 describes these levels.

Table 49. 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 Personal Communications. | 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 <i>Microsoft Windows Software Development Kit Guide to Programming</i> . | PS (presentation space) |

Personal Communications supports IBM3270 IBM5250 as Windows DDE server.

You can use the following 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 Personal Communications contains these associated data items:

| Item | Function |
|----------|---|
| Formats | Returns the list of clipboard formats (numbers) that Personal Communications is capable of rendering. |
| Status | Returns information about the status of each Personal Communications session. |
| SysCon | Returns the level of Personal Communications support and other system related values. |
| SysItems | Returns the list of data items that are available when connected to the Personal Communications system topic. |
| Topics | Returns the list of Personal Communications 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

Table 50 lists the DDE functions that are available for use with Personal Communications.

Table 50. DDE Functions in a 16-Bit Environment

| Function | PC/3270 Windows | PC400 |
|-------------------------------|--------------------|-------|
| "Find Field" on page 360 | Yes | Yes |
| "Get Keystrokes" on page 361 | Yes | Yes |
| "Get Mouse Input" on page 362 | Yes | Yes |

Table 50. DDE Functions in a 16-Bit Environment (continued)

| Function | PC/3270 Windows | PC400 |
|--|--------------------|-------|
| "Get Number of Close Requests" on page 365 | Yes | Yes |
| "Get Operator Information Area" on page 366 | Yes | Yes |
| "Get Partial Presentation Space" on page 367 | Yes | Yes |
| "Get Presentation Space" on page 369 | Yes | Yes |
| "Get Session Status" on page 371 | Yes | Yes |
| "Get System Configuration" on page 372 | Yes | Yes |
| "Get System Formats" on page 373 | Yes | Yes |
| "Get System Status" on page 374 | Yes | Yes |
| "Get System SysItems" on page 375 | Yes | Yes |
| "Get System Topics" on page 376 | Yes | Yes |
| "Get Trim Rectangle" on page 377 | Yes | Yes |
| "Initiate Session Conversation" on page 377 | Yes | Yes |
| "Initiate Structured Field Conversation" on page 378 | Yes | No. |
| "Initiate System Conversation" on page 379 | Yes | Yes |
| "Put Data to Presentation Space" on page 379 | Yes | Yes |
| "Search for String" on page 380 | Yes | Yes |
| "Send Keystrokes" on page 381 | Yes | Yes |
| "Session Execute Macro" on page 382 | Yes | Yes |
| "Set Cursor Position" on page 388 | Yes | Yes |
| "Set Mouse Intercept Condition" on page 390 | Yes | Yes |
| "Set Presentation Space Service Condition" on page 392 | Yes | Yes |
| "Set Session Advise Condition" on page 393 | Yes | Yes |
| "Set Structured Field Service Condition" on page 394 | Yes | No |
| "Start Close Intercept" on page 395 | Yes | Yes |
| "Start Keystroke Intercept" on page 397 | Yes | Yes |
| "Start Mouse Input Intercept" on page 398 | Yes | Yes |
| "Start Read SF" on page 401 | Yes | No |
| "Start Session Advise" on page 402 | Yes | Yes |
| "Stop Close Intercept" on page 403 | Yes | Yes |
| "Stop Keystroke Intercept" on page 404 | Yes | Yes |
| "Stop Mouse Input Intercept" on page 405 | Yes | Yes |
| "Stop Read SF" on page 405 | Yes | No |
| "Stop Session Advise" on page 406 | Yes | Yes |
| "Terminate Session Conversation" on page 407 | Yes | Yes |
| "Terminate Structured Field Conversation" on page 407 | Yes | No |
| "Terminate System Conversation" on page 407 | Yes | Yes |
| "Write SF" on page 408 | Yes | No |

Refer to "Summary of DDE Functions in a 16-Bit Environment" on page 420 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 Personal Communications starts to search a target field.
type Target field type. The field type are:

| Type | Meaning |
|----------|--|
| bb or Tb | This field. |
| Pb | The previous field, either protected or unprotected. |
| Nb | 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.

Personal Communications Response

Personal Communications 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 Personal Communications 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

Personal Communications 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.

Personal Communications Response

Personal Communications 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 Personal Communications 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

Personal Communications 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_DSPTXT
    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 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.

Personal Communications Response

Personal Communications 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 Personal Communications 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, Personal Communications 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 PPos[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) |

| Parameter Name | Meaning | Value |
|----------------|---|---|
| PSRowPos | Row number of the position where mouse was clicked | 0 ... (PSRows - 1) |
| PSColPos | Column number of the position where mouse was clicked | 0 ... (PSCols - 1) |
| PSSize | Size of the presentation space | |
| PSRows | Number of rows of presentation space | |
| PSCols | Number of columns of presentation space | |
| ButtonType | Type of the clicked mouse button | L Left button M Middle button R Right button |
| ClickType | Type of clicking | S Single click D Double click |
| ClickString | Retrieved string to which the mouse pointed | A character string terminated with a '\0' |
| Tab1-8 | A tab character for delimiter | '\t' |

If the format is CF_DSPTTEXT, Personal Communications returns the mouse input information in the following format:

```
typedef struct tagMOUSE_CF_DSPTTEXT
{
    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_DSPTTEXT
    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_DSPTTEXT, FAR *lpMOUSE_CF_DSPTTEXT;
```

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 | |

| Parameter Name | Meaning | Value |
|----------------|---|---|
| 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.

Personal Communications Response

Personal Communications 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 Personal Communications 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

Personal Communications 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.

Personal Communications Response

Personal Communications 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 Personal Communications 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

Personal Communications 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.

Personal Communications Response

Personal Communications 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 Personal Communications 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

Personal Communications returns the part of the presentation space in the format specified in the **Get Partial Presentation Space** request.

If the format is CF_DSPTEXT, Personal Communications 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_DSPTTEXT, FAR *lpEPS_CF_DSPTTEXT;

```

```

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.

```

lppls = (lp_EPS_CF_DSPTTEXT) lpEPS_CF_DSPTTEXT->PSData
        + lpEPS_CF_DSPTTEXT->uPSOffset;
lpplsfields = (lpPSFIELDS) lpEPS_CF_DSPTTEXT->PSData
              + lpEPS_CF_DSPTTEXT->uFieldOffset;

```

If the format is CF_TEXT, Personal Communications 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 PSENGTH[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.

Personal Communications Response

Personal Communications 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 Personal Communications 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

Personal Communications returns the presentation space in the format specified in the **Get Presentation Space** request.

If the format is CF_DSPTEXT, Personal Communications returns the presentation space in the following format:

```

typedef struct tagPS_CF_DSPTTEXT
{
    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_DSPTTEXT, FAR *lpPS_CF_DSPTTEXT;

```

```

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.

```

lppps = (lp_PS_CF_DSPTTEXT) lpPS_CF_DSPTTEXT->PSData
        + lpPS_CF_DSPTTEXT->uPSOffset;
lppsfields = (lpPSFIELDS) lpPS_CF_DSPTTEXT->PSData
             + lpPS_CF_DSPTTEXT->uFieldOffset;

```

If the format is CF_TEXT, Personal Communications 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;
lpss_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.

Personal Communications Response

Personal Communications 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 Personal Communications 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

Personal Communications 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. |

| Fields | Returned values | Description |
|-----------------------------------|----------------------|---|
| 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. |

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 Personal Communications.

Get System Configuration

| | | |
|------|------|-----|
| 3270 | 5250 | VT |
| Yes | Yes | Yes |

The **Get System Configuration** function returns the level of Personal Communications 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.

Personal Communications Response

Personal Communications 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 Personal Communications 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

Personal Communications 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 Personal Communications |
| Level | NN | The level of Personal Communications |
| 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 Personal Communications. The client application sends the following message to retrieve the format list supported by Personal Communications:

```
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.

Personal Communications Response

Personal Communications 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 Personal Communications:

- CF_TEXT
- CF_DSPTEXT

If Personal Communications 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.

Personal Communications Response

Personal Communications returns the status data item in CF_TEXT format in a DDE DATA message:

```
WM_DDE_DATA(hData, aSTATUS)
```


For each opened session, Personal Communications 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 Personal Communications. |
| Emulation Type | 3270, 5250 | The emulation type supported by Personal Communications. |
| Session Status | Closed, Invisible, Normal, Minimized, Maximized | The current status of the session's window. |

If Personal Communications 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 Systems

| | | |
|------|------|-----|
| 3270 | 5250 | VT |
| Yes | Yes | Yes |

Personal Communications supports the DDE system topic so that a client application can connect to the system topic and retrieve information about Personal Communications and the status of the sessions that Personal Communications is managing.

The **Get System SysItems** function returns the list of data items available in the Personal Communications 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.

Personal Communications Response

Personal Communications 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 Personal Communications:

- SysItems
- Topics
- Status
- Formats
- SysCon

If Personal Communications 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 Personal Communications. 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.

Personal Communications Response

Personal Communications 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 Personal Communications:

- System – System Topic
- SessionA – Session A Topic
- ⋮
- SessionZ – Session Z Topic

Note: 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 Personal Communications 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)
```

| 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.

Personal Communications Response

Personal Communications 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 Personal Communications 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 Personal Communications. 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.

Personal Communications Response

If a specific topic is selected and Personal Communications can support a conversation with the client application, Personal Communications acknowledges the INITIATE transaction with:

```
WM_DDE_ACK(aIBM327032, aSessionN)
```

If a topic is not selected (aSessionN = NULL), Personal Communications 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 an underscore (_), 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.

Personal Communications Response

If Personal Communications 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_DSPTXT
    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.

Personal Communications Response

Personal Communications 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, Personal Communications 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_DSPTXT.
aSTRING Identifies the search data item.

Personal Communications Response

Personal Communications 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 Personal Communications 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

Personal Communications 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_DSPTXT
    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_DSPTXT
    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.

Personal Communications Response

Personal Communications 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 Personal Communications cannot send the keystrokes to the host, Personal Communications returns a negative ACK message containing one of the following status codes in the low-order byte of the wStatus word. Otherwise, Personal Communications 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 Personal Communications.

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 Personal Communications commands. For a list of commands you can issue, see "Issuing Commands with the Session Execute Macro Function."

Personal Communications Response

If Personal Communications can process the command string, Personal Communications returns an ACK message containing positive status information to the client. If Personal Communications cannot perform the command string, Personal Communications 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.

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 *Administrator's Reference* 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 *Administrator's Reference* for more information about the transfer options.

SENDKEY Command

```
[SENDKEY(token,token)]
```

Sends keystrokes to Personal Communications, where:

```
token = text string|command|macro macroname
```

Note:

- 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 51. 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 |
| 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 |

Table 51. SENDKEY Command List (continued)

| Command Name | Token | PC/3270 | PC400 |
|----------------------------|---------------|---------|-------|
| 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 |
| 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 |

Table 51. SENDKEY Command List (continued)

| Command Name | Token | PC/3270 | PC400 |
|--|-----------------------|---------------|---------------|
| 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 | rtn | 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 |
| 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 time out 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_DSPTXT
    unsigned uSetCursorType;     // Cursor Set Type
    unsigned uSetCursor1;        // Cursor Row or PS Offset
    unsigned uSetCursor2;        // Cursor Col
} SETCURSOR, FAR *lpSETCURSOR;
```

Personal Communications 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.

Personal Communications Response

Personal Communications receives the cursor information and moves the cursor to the specified position in the PS. If the cursor is positioned successfully, Personal Communications 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). |
| 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, CF_TEXT or CF_DSPTEXT.
aMOUSE Identifies Mouse atom as the item.

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_DSPTXT
{
    unsigned    unused:12,        //
    unsigned    fRelease:1,      //
    unsigned    fReserved:3;     //
    int         cfFormat;        // Always CF_DSPTXT
    BOOL        bLeftButton;     //
    BOOL        bRightButton;    //
    BOOL        bMiddleButton;   //
    BOOL        bSingleClick;    //
    BOOL        bDoubleClick;    //
    BOOL        bRetrieveString; //
} SETMOUSE_CF_DSPTXT, FAR *lpSETMOUSE_CF_DSPTXT;

```

The following table shows the values in the parameters:

| Parameter Name | Meaning | Value |
|-----------------|---|--|
| bLeftButton | Enable or disable interception of the left mouse button | True Enable intercepting the left button False Disable intercepting the left button |
| bRightButton | Enable or disable interception of the right mouse button | True Enable intercepting the right button False Disable intercepting the right button |
| bMiddleButton | Enable or disable interception of the middle mouse button | True Enable intercepting the middle button False Disable intercepting the middle button |
| bSingleClick | Enable or disable interception of the single click | True Enable intercepting the single click False Disable intercepting the single click |
| bDoubleClick | Enable or disable interception of the double click | True Enable intercepting the double click False Disable intercepting the double click |
| bRetrieveString | Retrieve or do not retrieve the pointed string | True Retrieve the pointed string False Do not retrieve the pointed string |

Personal Communications Response

When receiving the **Set Mouse Intercept Condition** request, Personal Communications returns an ACK message if it can set the intercept condition to the specified status. Otherwise, a negative ACK message is returned to the client with one of the following return codes in the low-order byte of the wStatus field:

WM_DDE_ACK(wStatus, aMOUSE)

| Return Code | Explanation |
|-------------|--|
| 2 | A character in Condition parameter is not valid. |

| Return Code | Explanation |
|-------------|------------------------------------|
| 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.

Personal Communications Response

If Personal Communications can perform the **Set Presentation Space Service Condition** function, then Personal Communications returns an ACK message:

```
WM_DDE_ACK(wStatus, aEPSCOND)
```

If Personal Communications cannot perform the **Set Presentation Space Service Condition** function, then Personal Communications 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 402 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_DSPTXT
    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.

Personal Communications Response

If Personal Communications can perform the **Set Session Advise Condition** function, Personal Communications returns this ACK message:

```
WM_DDE_ACK(wStatus, aPSCOND)
```

If Personal Communications cannot perform the **Set Session Advise Condition** function, then Personal Communications 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_DSPTTEXT
    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.

Personal Communications Response

Personal Communications 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 Personal Communications.

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.

Personal Communications Response

Personal Communications 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 Personal Communications. 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.

Personal Communications Response

Personal Communications 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, Personal Communications returns the mouse input information in the following format:

```
typedef struct tagMOUSE_CF_TEXT
{
    unsigned    unused:12,        // **** Unused ****
    unsigned    fRespons:1,      // TRUE = DDE_REQUEST response
    unsigned    fRelease:1,      // TRUE = Client frees this data
    unsigned    reserved:1,      // **** Reserved ****
    unsigned    fAckReq:1;       // TRUE = Client returns DDE_ACK
    int         cfFormat;        // CF_TEXT
    unsigned char PSPos[4];      // PS position
    unsigned char Tab1[1];       // TAB character
    unsigned char PSRowPos[4];   // PS row position
    unsigned char Tab2[1];       // TAB character
    unsigned char PSColPos[4];   // PS columns position
    unsigned char Tab3[1];       // TAB character
    unsigned char PSSize[4];     // Size of the PS
    unsigned char Tab4[1];       // TAB character
    unsigned char PSRows[4];     // PS number of rows
    unsigned char Tab5[1];       // TAB character
    unsigned char PSCols[4];     // PS number of columns
    unsigned char Tab6[1];       // TAB character
    unsigned char ButtonType[1]; // Pressed button type
    unsigned char Tab7[1];       // TAB character
    unsigned char ClickType[1];  // Click type
    unsigned char Tab8[1];       // TAB character
    unsigned char ClickString[1]; // Retrieved string
} MOUSE_CF_TEXT, FAR *lpMOUSE_CF_TEXT;
```

The following table shows the values in the parameters:

| Parameter Name | Meaning | Value |
|----------------|---|---|
| PSPos | PS offset of the position where mouse was clicked | 0 ... (PSSize - 1) |
| PSRowPos | Row number of the position where mouse was clicked | 0 ... (PSRows - 1) |
| PSColPos | Column number of the position where mouse was clicked | 0 ... (PSCols - 1) |
| PSSize | Presentation space size | |
| PSRows | Number of presentation space rows | |
| PSCols | Number of presentation space columns | |
| ButtonType | Type of clicked mouse button | L Left button M Middle button R Right button |
| ClickType | Type of clicking | S Single click D Double click |

| Parameter Name | Meaning | Value |
|----------------|---|---|
| ClickString | Retrieved string to which the mouse pointed | A character string terminated with a '\0' |
| Tab1-8 | A tab character for delimiter | '\t' |

hData If the format is CF_DSPTTEXT, Personal Communications returns the mouse input information in the following format:

```
typedef struct tagMOUSE_CF_DSPTTEXT
{
    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_DSPTTEXT
    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_DSPTTEXT, FAR *lpMOUSE_CF_DSPTTEXT;
```

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 Personal Communications.

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_DSPTXT
    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 Personal Communications 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 393.

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`.

Personal Communications Response

Personal Communications 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 Personal Communications. 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.

Personal Communications Response

If Personal Communications can perform the DDE_UNADVISE, Personal Communications returns an ACK message containing positive status information to the client:

```
WM_DDE_ACK(wStatus, aCLOSE)
```

If Personal Communications cannot perform the DDE_UNADVISE, Personal Communications 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.

Personal Communications Response

If Personal Communications can perform the DDE_UNADVISE, Personal Communications returns an ACK message containing positive status information to the client:

```
WM_DDE_ACK(wStatus, aKEYS)
```

If Personal Communications cannot perform the DDE_UNADVISE, Personal Communications 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. |

| Return Code | Explanation |
|-------------|--------------------------|
| 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.

Personal Communications Response

If Personal Communications can perform the DDE_UNADVISE, Personal Communications returns an ACK message containing positive status information to the client:

```
WM_DDE_ACK(wStatus, aMOUSE)
```

If Personal Communications cannot perform the DDE_UNADVISE, Personal Communications 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,
             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 Personal Communications 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.

Personal Communications Response

If Personal Communications can perform the DDE_UNADVISE, Personal Communications returns an ACK message containing positive status information to the client:

```
WM_DDE_ACK(wStatus, aItem)
```

If Personal Communications cannot perform the DDE_UNADVISE, Personal Communications 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 Personal Communications 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) );
```

Personal Communications Response

Personal Communications 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) );
```

Personal Communications Response

Personal Communications acknowledges the terminate command with this message:

```
WM_DDE_TERMINATE
```

When the user closes a Personal Communications session, any global memory blocks that were allocated by Personal Communications 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_DSPTXT  
    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. |

DDE Menu Item API in a 16-Bit Environment

Personal Communications 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.

Personal Communications supports two kinds of DDE conversation. One is Personal Communications, which acts as a DDE menu client application, and the other is Personal Communications, 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.

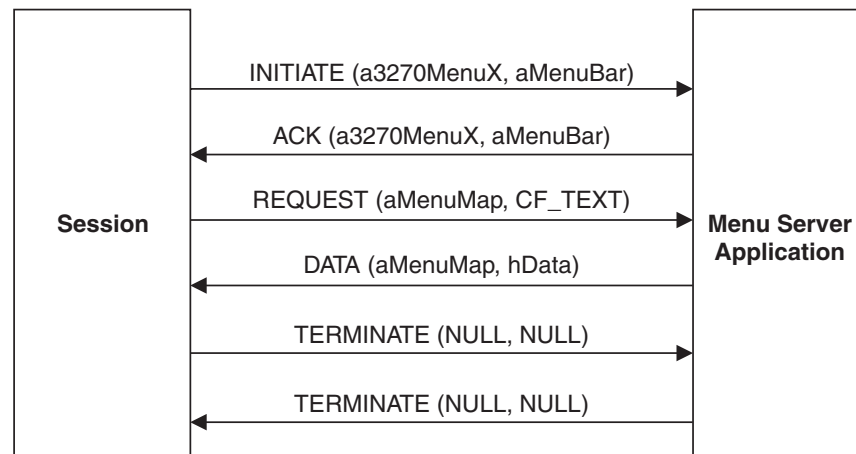


Figure 8. DDE Menu Server Conversation

The following data hierarchy details the menu map Personal Communications 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, Personal Communications 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, Personal Communications 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 Personal Communications the process is complete. Personal Communications will then reenable the window for the user.

DDE Menu Server, 32-Bit

To add, delete, and change menu items, the Figure 9 on page 410 must take place between the session and a DDE menu client application.

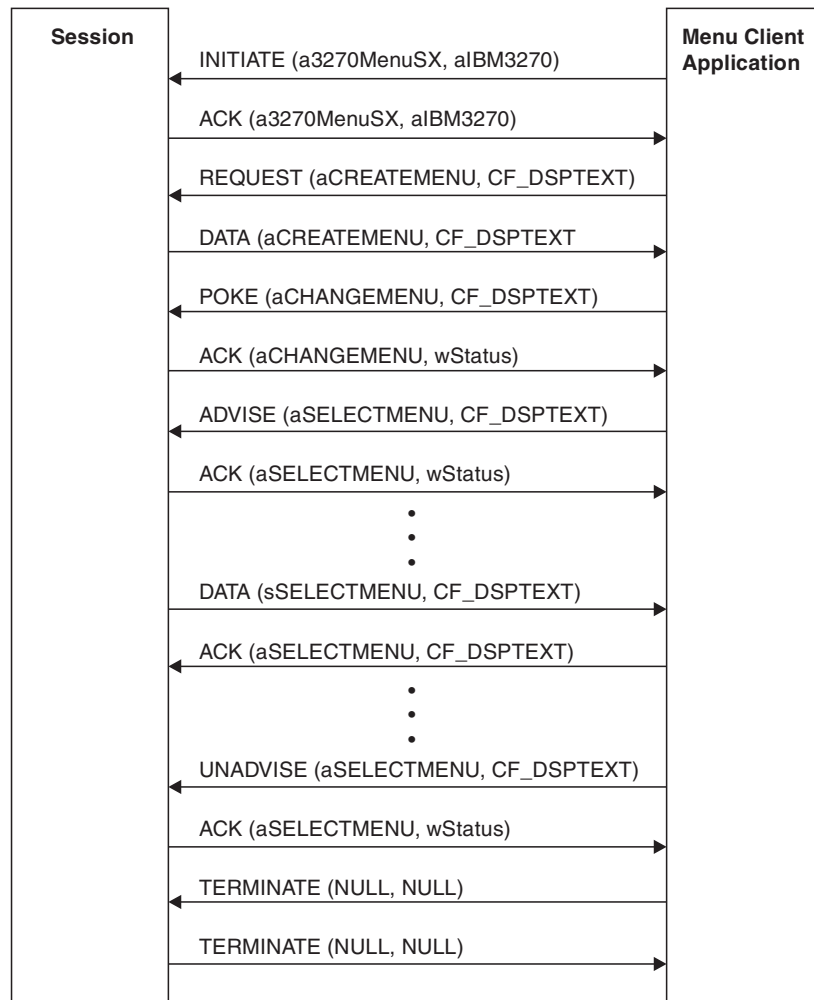


Figure 9. DDE Menu Client Conversation

When the user selects a menu item from the new menu, Personal Communications will send a DDE DATA with aSELECTMENU as the item. When Personal Communications sends DDE DATA to the client application, Personal Communications 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 Personal Communications the process is complete. Personal Communications will then reenable the window for the user.

DDE Menu Functions in a 16-Bit Environment

This section describes the DDE Menu Item API functions that are available for use with Personal Communications. PC/3270 Windows mode and PC400 provide all of the listed functions.

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_DSPTXT  
    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 Personal Communications 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 Personal Communications when **Create Menu Item** function is executed.

| | |
|-------------------|--|
| wFlags | The following options can be set: |
| | <pre> MF_CHECKED // Places a check mark 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 check mark // next to the item (default). </pre> |
| 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 Personal Communications 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. |
| | <p>MF_BYPOSITION Specifies the position of the existing menu item. The first item in the menu is at position zero.</p> <p>MF_BYCOMMAND Specifies the command ID of the existing menu item.</p> |
| 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 Personal Communications 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 check mark 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 check mark
                 // 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 Personal Communications 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       // MF_BYPOSITION is set.
                      // 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 Personal Communications 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 Personal Communications 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 check mark 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 check mark 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 Personal Communications 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.

```

Personal Communications Response

Personal Communications receives the requests to change a menu and processes them. If the requests cannot be accepted, Personal Communications returns a negative ACK message containing one of the following status codes in the low-order byte of the wStatus word. Otherwise, Personal Communications 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 Personal Communications 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.

Personal Communications Response

Personal Communications returns the handle of the newly created menu item in a dde data message if the Personal Communications can create a menu item.

WM_DDE_DATA(hData, aCREATEMENU)

or
 WM_DDE_ACK(wStatus, 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;
```

If Personal Communications 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 Personal Communications. 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.

Personal Communications Response

If Personal Communications can support a conversation with the client application, Personal Communications 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_DSPTXT  
} OPTIONS, FAR *lpOPTIONS;
```

aSELECTMENU Identified a menu advise as the item.

Personal Communications Response

Personal Communications receives the **Start Menu Advise** and returns an ACK message if it can start the function.

```
WM_DDE_ACK(wStatus, aSELECTMENU)
```

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. |

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_DSPTXT
    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 Personal Communications.

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.

Personal Communications Response

If Personal Communications can perform the DDE_UNADVISE, Personal Communications returns an ACK message containing positive status information to the client:

```
WM_DDE_ACK(wStatus, aCLOSE)
```

If Personal Communications cannot perform the DDE_UNADVISE, Personal Communications 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 Personal Communications 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) );
```

Personal Communications Response

Personal Communications acknowledges the terminate command with this message:

WM_DDE_TERMINATE

Summary of DDE Functions in a 16-Bit Environment

Table 52 lists the DDE functions that can be used with Personal Communications. The table lists the name of the DDE function, the command the client sends to Personal Communications, and the values that can be used for the variables in the client command.

Table 52. Summary of DDE Functions in a 16-Bit Environment

| Function name | Client command |
|------------------|---|
| Change Menu Item | PostMessage(hServerWnd, WM_DDE_POKE, hClientWnd, MAKELONG(hData, aCHANGEMENU)); |
| | hData = Handle to a global memory object |
| Create Menu Item | PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELONG(cfFormat, aCREATEMENU)); |
| | cfFormat = CF_DSPTEXT |
| 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 |

Table 52. Summary of DDE Functions in a 16-Bit Environment (continued)

| Function name | Client command |
|--------------------------------|---|
| Get Mouse Input | PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELONG(cfFormat, aMOUSE)); |
| | cfFormat = CF_TEXT CF_DSPTEXT |
| Get Number of Close Requests | PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELONG(cfFormat, aCLOSE)); |
| | cfFormat = CF_DSPTEXT |
| Get Operator Information Area | PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELONG(cfFormat, aOIA)); |
| | cfFormat = CF_DSPTEXT |
| Get Partial Presentation Space | PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELONG(cfFormat, aEPS)); |
| | cfFormat = CF_TEXT CF_DSPTEXT |
| Get Presentation Space | PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELONG(cfFormat, aPS)); |
| | cfFormat = CF_TEXT CF_DSPTEXT |
| Get Session Status | PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELONG(cfFormat, aSSTAT)); |
| | cfFormat = CF_TEXT |
| Get System Configuration | PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELONG(cfFormat, aSYSCON)); |
| | cfFormat = CF_TEXT |
| 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 Systems | PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELONG(cfFormat, aSYSTEMS)); |
| | cfFormat = CF_TEXT |

Table 52. Summary of DDE Functions in a 16-Bit Environment (continued)

| Function name | Client command |
|--|--|
| Get System Topics | PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELONG(cfFormat, aTOPICS)); |
| | cfFormat = CF_TEXT |
| Get Trim Rectangle | PostMessage(hServerWnd, WM_DDE_REQUEST, hClientWnd, MAKELONG(cfFormat, aTRIMRECT)); |
| | cfFormat = CF_TEXT |
| Initiate Menu Conversation | PostMessage(hServerWnd, WM_DDE_INITIATE, hClientWnd, MAKELONG(aIBM327032, SN)); |
| | N = a session letter A through Z |
| 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 |

Table 52. Summary of DDE Functions in a 16-Bit Environment (continued)

| Function name | Client command |
|--|---|
| Set Cursor Position | PostMessage(hServerWnd, WM_DDE_POKE, hClientWnd, MAKELONG(hData, aSETCURSOR)); |
| | hData = Handle to a global memory object |
| Set Mouse Intercept Condition | PostMessage(hServerWnd, WM_DDE_POKE, hClientWnd, MAKELONG(hData, aMOUSE)); |
| | hData = Handle to a global memory object |
| Set Presentation Space Service Condition | PostMessage(hServerWnd, WM_DDE_POKE, hClientWnd, MAKELONG(hData, aEPSCOND)); |
| | hData = Handle to a global memory object |
| Set Session Advise Condition | PostMessage(hServerWnd, WM_DDE_POKE, hClientWnd, MAKELONG(hData, aPSCOND)); |
| | hData = Handle to a global memory object |
| Set Structured Field Service Condition | PostMessage(hServerWnd, WM_DDE_POKE, hClientWnd, MAKELONG(hData, aSFCOND)); |
| | hData = Handle to a global memory object |
| Start Close Intercept | SendMessage(hServerWnd, WM_DDE_ADVISE, hClientWnd, MAKELONG(hOptions, aCLOSE)); |
| | hOptions = Handle to a global memory object |
| Start Keystroke Intercept | SendMessage(hServerWnd, WM_DDE_ADVISE, hClientWnd, MAKELONG(hOptions, aKEYS)); |
| | hOptions = Handle to a global memory object |
| Start Menu Advise | PostMessage(hServerWnd, WM_DDE_ADVISE, hClientWnd, MAKELONG(hOptions, aSELECTMENU)); |
| | 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 |

Table 52. Summary of DDE Functions in a 16-Bit Environment (continued)

| Function name | Client command |
|---|---|
| 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)); |
| Start Mouse Input Intercept | PostMessage(hServerWnd, WM_DDE_ADVISE, hClientWnd, MAKELONG(hOptions, aMOUSE)); |
| | hOptions = Handle to a global memory object |
| Stop Menu Advise | PostMessage(hServerWnd, WM_DDE_UNADVISE, hClientWnd, MAKELONG(NULL, aSELECTMENU)); |
| 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 Menu 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 |

Appendix F. 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.

On Windows platforms, REXX applications require the use of IBM Object REXX for Windows.

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 with 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 available, 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 required. Prerequisite calls for all REXX EHLLAPI functions are listed under “Summary of Prerequisite Calls for Functions” on page 426.

Supplied Syntax Illustrates the syntax of the function call. Refer to “Overview of REXX EHLLAPI Function Calls and Return Values” 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 <i>result</i> . 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 53 lists the prerequisite calls for each REXX EHLLAPI function. Prerequisite calls are required when you use an application program.

Table 53. 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_Str | 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 |
| Lock_PMSVC | Connect_PM |
| Lock_PS | Connect |
| Pause | None |
| Query_Close_Intercept | Start_Close_Intercept |
| Query_Cursor_Pos | Connect |
| Query_Emulator_Status | None |
| Query_Field_Attr | Connect |

Table 53. Prerequisite Calls for Functions (continued)

| Function | Prerequisite Calls |
|---------------------------|---------------------------|
| Query_Host_Update | Start_Host_Notify |
| Query_Session_List | None |
| Query_Session_Status | None |
| Query_Sessions | None |
| Query_System | None |
| Query_Window_Coord | Connect_PM |
| Query_Workstation_Profile | None |
| 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_Communication | None |
| Start_Host_Notify | None |
| Start_Keystroke_Intercept | None |
| Start_Session | None |
| Stop_Close_Intercept | Start_Close_Intercept |
| Stop_Communication | None |
| Stop_Host_Notify | Start_Host_Notify |
| Stop_Keystroke_Intercept | Start_Keystroke_Intercept |
| Stop_Session | None |
| Wait | Connect |

Summary of EHELLAPI and REXX EHELLAPI Functions

Table 54 lists each EHELLAPI and REXX EHELLAPI function.

Table 54. EHELLAPI and REXX EHELLAPI Functions

| EHELLAPI | REXX EHELLAPI |
|----------------------------------|--------------------|
| 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 |

Table 54. EHLAPI and REXX EHLAPI Functions (continued)

| EHLAPI | REXX EHLAPI |
|---|---|
| CONVERT POSITION or CONVERT ROWCOL (99) | Convert_Pos |
| COPY FIELD TO STRING (34) | Copy_Field_To_String |
| COPY OIA (13) | Copy_OIA |
| 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 |

Table 54. EHLAPI and REXX EHLAPI Functions (continued)

| EHLAPI | REXX EHLAPI |
|--------------------------------|---------------------------|
| 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 |
| pcsStartSession | Start_Session |
| pcsStopSession | Stop_Session |
| pcsConnectSession | Start_Communication |
| pcsDisconnectSession | Stop_Communication |
| pcsQuerySessionList | Query_Session_List |
| pcsQueryEmulatorStatus | Query_Emulator_Status |
| pcsQueryWorkstationProfile | Query_Workstation_Profile |

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.

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 <code>Change_Switch_Name</code> 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

There are no prerequisite calls for this function.

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.

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

There are no prerequisite calls for this function.

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

There are no prerequisite calls for this function.

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 <i>c</i> is the column number and <i>r</i> 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 <i>pos</i> , or a <i>pos</i> that is not valid was specified. |
| data | <p>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.</p> <p>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.</p> |

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 Parameters

There are no supplied parameters for this function.

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 48. |

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 Parameters

There are no supplied parameters for this function.

Returned Parameters

| Return Code | Explanation |
|-------------|--|
| '' | Null. Not connected or other error. |
| data | <p>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.</p> <p>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.</p> |

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 <code>Set_Session_Parms</code> 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 <i>pos</i> 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 <code>Copy_Str_To_PS</code> 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 <code>pos</code> parameter was not valid. |
| 9 | A system error occurred. |

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 Parameters

There are no supplied parameters for this function.

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

The **Find_Field_Len** function returns the length of the target field along with the attributes specified by the *search_option* parameter.

Prerequisite Calls

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 'Tb' | Current field (the field where the cursor is located). |
| 'Nb' | Next field, either protected or unprotected. |
| 'Pb' | 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

The `Find_Field_Pos` function returns the location of the target field with the attributes specified by the `search_option` parameter.

Prerequisite Calls

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 'Tb' | The current field (the field where the cursor is located). |
| 'Nb' | The next field, either protected or unprotected. |
| 'Pb' | 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

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.

Prerequisite Calls

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 <i>session_id</i> . |
| 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 137. |

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

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

Prerequisite Calls

`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 <code>Connect_PM</code> 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

The **Intercept_Status** function informs the session when a keystroke obtained through the **Get_Key** function was accepted or rejected.

Prerequisite Calls

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

This function locks or unlocks the presentation space window.

Prerequisite Calls

Connect_PM

Supplied Syntax

The syntax for the Lock_PMSVC function is as follows:

```
EHLLAPI( '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 | 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

The **Lock_PS** function locks or unlocks the presentation space.

Prerequisite Calls

Connect

Supplied Syntax

The syntax for the **Lock_PS** function is as follows:

```
EHLLAPI( '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 | 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

The **Pause** function causes a timed pause of $n \frac{1}{2}$ -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).

Prerequisite Calls

There are no prerequisite calls for this function.

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 455 for more information. |

Query_Close_Intercept

The **Query_Close_Intercept** function determines if a close request was started from the session.

Prerequisite Calls

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

The **Query_Cursor_Pos** function returns the cursor position for the currently connected session.

Prerequisite Calls

Connect

Supplied Syntax

The syntax for the **Query_Cursor_Pos** function is as follows:

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

Supplied Parameters

There are no supplied parameters for this function.

Returned Parameters

| Return Code | Explanation |
|-------------|--|
| 0 | Your program is not currently connected to the host session. |
| data | Cursor position. |

Query_Emulator_Status

The `Query_Emulator_Status` function returns the status of the specified host session.

Prerequisite Calls

There are no prerequisite calls for this function.

Supplied Syntax

The syntax for the `Query_Emulator_Status` function is as follows:

```
HLLAPI( 'Query_emulator_status', session_id )
```

Supplied Parameters

Function name: `Query_Emulator_Status`
session_id: Specifies the session letter (A-Z) of the session to be queried.

Returned Parameters

| Return Code | Explanation |
|-------------|--|
| ' ' | Null. Error has occurred. |
| 1 | Session started. |
| 2 | Session started and connection to the host requested. |
| 3 | Session started, connection requested, and API is enabled for the session. |

Query_Field_Attr

The `Query_Field_Attr` function returns the hexadecimal representation of the field attribute in the currently connected session.

Prerequisite Calls

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

The **Query_Host_Update** function determines if the OIA or presentation space for the session has been updated.

Prerequisite Calls

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. |
| 44 | Printing has completed in the printer session. |

Query_Session_List

The `Query_Session_List` function returns a 2-byte entry of each current host session.

Prerequisite Calls

There are no prerequisite calls for this function.

Supplied Syntax

The syntax for the `Query_Session_List` function is as follows:

```
HLLAPI( 'Query_session_list', )
```

Supplied Parameters

There are no supplied parameters for this function.

Returned Parameters

| Return Code | Explanation | |
|-------------|--------------------------|--|
| ' ' | Null. Not connected. | |
| 0 | No sessions are started. | |
| data | Position | Definition |
| | 1 | Short session ID. |
| | 2 | One of the following values: 1 Session started. 0 Session started and connection to the host requested. 3 Session started, connection requested, and API is enabled for the session. |

Query_Session_Status

The `Query_Session_Status` function returns various status information from the host session, or from the currently connected session if `session_id` is blank.

Prerequisite Calls

There are no prerequisite calls for this function.

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: <ul style="list-style-type: none">• 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

The **Query_Sessions** function returns either a 12-byte description of each configured session or a null (") if an error occurs.

Prerequisite Calls

There are no prerequisite calls for this function.

Supplied Syntax

The syntax for the **Query_Sessions** function is as follows:

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

Supplied Parameters

There are no supplied parameters for this function.

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 112.) | |

Note: After you parse the last field (pssize) from the string to obtain its decimal values, use `c2d(reverse(x))`.

Query_System

The **Query_System** function returns either a 35-byte system configuration string or a null (") if an error occurs.

Prerequisite Calls

There are no prerequisite calls for this function.

Supplied Syntax

The syntax for the **Query_System** function is as follows:

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

Supplied Parameters

There are no supplied parameters for this function.

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: <ul style="list-style-type: none">• V = VGA• H = XGA• U = unknown | |
| 33-35 | Reserved. | |

Query_Window_Coord

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.

Prerequisite Calls

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 |

Query_Workstation_Profile

The **Query_Workstation_Profile** function returns the profile name that was used to start the specified host session.

Prerequisite Calls

There are no prerequisite calls for this function.

Supplied Syntax

The syntax for the **Query_Workstation_Profile** function is as follows:

```
HLLAPI( 'Query_workstation_profile', session_id )
```

Supplied Parameters

Function name: **Query_Workstation_Profile**
session_id: Specifies the session letter (A-Z) of the session to be queried.

Returned Parameters

| Return Code | Explanation |
|-------------|--|
| '' | Null. Error has occurred. |
| data | The name of the workstation profile used to start the session. |

Receive_File

The **Receive_File** function is used to transfer a file from the host session to the workstation session.

Note: Do not terminate the Receive program while file transfer is in progress; otherwise you will receive an error message.

Prerequisite Calls

There are no prerequisite calls for this function.

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 121.

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 <i>OS/2 Control Program Programming Reference</i> . |

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

The **Release** function unblocks the connected display session keyboard.

Prerequisite Calls

Connect

Supplied Syntax

The syntax for the **Release** function is as follows:

```
HLLAPI( 'Release' )
```

Supplied Parameters

There are no supplied parameters for this function.

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

The **Reserve** function blocks the currently connected session from user input until either a **Release** or a **Disconnect** function is executed.

Prerequisite Calls

Connect

Supplied Syntax

The syntax for the **Reserve** function is as follows:

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

Supplied Parameters

There are no supplied parameters for this function.

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

The **Reset_System** function reinitializes the session parameters (set by the **Set_Session_Parms** function) to their defaults and disconnects from all connected resources.

Prerequisite Calls

There are no prerequisite calls for this function.

Supplied Syntax

The syntax for the **Reset_System** function is as follows:

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

Supplied Parameters

There are no supplied parameters for this function.

Returned Parameters

| Return Code | Explanation |
|-------------|--|
| 0 | The Reset_System function was successful. |
| 9 | A system error occurred. |

Search_Field

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.

Prerequisite Calls

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 <i>string</i> was not found or the session was not connected. |
| data | The position of the <i>string</i> 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

The `Search_PS` function searches the host presentation space for a particular string.

Prerequisite Calls

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 <i>string</i> was not found or the session was not connected. |
| data | Position of the <i>string</i> 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

The **Send_File** function transfers a file from the personal computer session where EHLLAPI is running to a host session.

Note: Do not terminate the Send program while file transfer is in progress; otherwise, you will receive an error message.

Prerequisite Calls

There are no prerequisite calls for this function.

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 133.

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 <i>OS/2 Control Program Programming Reference</i> |

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

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.

Prerequisite Calls

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 137.

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

The `Set_Cursor_Pos` function positions the cursor at the specified target field within the currently connected host presentation space.

Prerequisite Calls

Connect

Supplied Syntax

The syntax for the `Set_Cursor_Pos` function is as follows:

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

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

The `Set_Session_Parms` function sets the current session parameters.

Prerequisite Calls

There are no prerequisite calls for this function.

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

The `Set_Window_Status` function changes the window status of a session.

Prerequisite Calls

`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 <i>option1</i> : 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 <i>num1</i> and <i>num2</i> represent the decimal position of the lower left corner of the new window position. |
| 'S' | Where <i>num1</i> and <i>num2</i> 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 <code>Set_Window_Status</code> 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

The **Start_Close_Intercept** function intercepts close requests for the host session.

Prerequisite Calls

There are no prerequisite calls for this function.

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_Communication

The **Start_Communication** function starts the communications with the host session for the specified *session_id*. This call is equivalent to doing a 'Communications->Connect' from the emulator window.

Prerequisite Calls

There are no prerequisite calls for this function.

Supplied Syntax

The syntax for the **Start_Communication** function is as follows:

```
HLLAPI( 'Start_communication', session_id )
```

Supplied Parameters

Function name: **Start_Communication**

session_id: The single character short-name of the session.

Returned Parameters

| Return Code | Explanation |
|-------------|---|
| 0 | The Start_Communication connection was requested successfully. |
| 1 | An incorrect session ID was specified. |
| 2 | The specified session has not been started. |

Start_Host_Notify

The **Start_Host_Notify** function determines if the designated host presentation space or operator information area has been updated.

Prerequisite Calls

There are no prerequisite calls for this function.

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.
E Asks for notification of completion during a printer 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

The **Start_Keystroke_Intercept** function filters any keystrokes sent to the session specified by the *session_id* parameter.

Prerequisite Calls

There are no prerequisite calls for this function.

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. |

Start_Session

The **Start_Session** function starts a host session using the specified workstation profile and optional *session_id* and start options.

Prerequisite Calls

There are no prerequisite calls for this function.

Supplied Syntax

The syntax for the **Start_Session** function is as follows:

```
HLLAPI( 'Start_session', profile_name, option[,  
        session_id] )
```

Supplied Parameters

Function name: **Start_Session**

session_id: An optional parameter, specifies the session letter (A-Z) to be associated with the session to be started. If null, the next available session letter will be used.

profile_name: The filename of the workstation profile to be started. The path can be included but is optional.

option: See the following table:

| Option | Explanation |
|--------|--|
| 'V' | Start the session with the window visible. |
| 'I' | Start the session with the window invisible. |
| 'X' | Start the session with the window maximized. |
| 'N' | Start the session with the window minimized. |

Returned Parameters

| Return Code | Explanation |
|-------------|---|
| 0 | The Start_Session function was successful. |
| 1 | An incorrect session ID was specified. |
| 2 | The specified session ID is already in use. |
| 3 | The workstation profile name is invalid. |
| 4 | An invalid operation was specified. |
| 9 | A system error occurred. |

Stop_Close_Intercept

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*.

Prerequisite Calls

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_Communication

The **Stop_Communication** function stops the communications with the host session for the specified *session_id*. This call is equivalent to doing a Communications->Disconnect from the emulator window.

Prerequisite Calls

There are no prerequisite calls for this function.

Supplied Syntax

The syntax for the **Stop_Communication** function is as follows:

```
HLLAPI( 'Stop_communication', session_id )
```

Supplied Parameters

Function name: **Stop_Communication**

session_id: The single character short-name of the session.

Returned Parameters

| Return Code | Explanation |
|-------------|---|
| 0 | The Stop_Communication disconnection was requested successfully. |
| 1 | An incorrect session ID was specified. |
| 2 | The specified session has not been started. |

Stop_Host_Notify

The **Stop_Host_Notify** function prevents the **Start_Host_Notification** function from determining if the host session identifier has been updated.

Prerequisite Calls

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

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.

Prerequisite Calls

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. |

Stop_Session

The **Stop_Session** function stops the host session specified.

Prerequisite Calls

There are no prerequisite calls for this function.

Supplied Syntax

The syntax for the **Stop_Session** function is as follows:

```
HLLAPI( 'Stop_session', session_id, save_option)
```

Supplied Parameters

Function name: **Stop_Session**

session_id: Specifies the session letter (A-Z) of the session to be stopped.

save_option: See the following table:

| Option | Explanation |
|--------|--|
| 'D' | Use the default profile save option as specified in the profile. |
| 'S' | Save the profile on exit. |
| 'N' | Do not save the profile on exit. |

Returned Parameters

| Return Code | Explanation |
|-------------|--|
| 0 | The Stop_Session function was successful. |
| 1 | An incorrect session ID was specified. |
| 2 | The specified session ID has not been started. |

Wait

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.

Prerequisite Calls

Connect

Supplied Syntax

The syntax for the **Wait** function is as follows:

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

Supplied Parameters

There are no supplied parameters for this function.

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

Several EHLLAPI features are not applicable to the REXX EHLLAPI environment. Structured fields of the EHLLAPI features are not supported by REXX EHLLAPI.

Sample Programs

Sample programs demonstrating the use of REXX EHLLAPI features are included on the CD-ROM.

The first sample program (QTIME.COMD) 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.COMD) 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.

Appendix G. Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. 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. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

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
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM documentation or non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those documents or Web sites. The materials for those documents or Web sites are not part of the materials for this IBM product and use of those documents or Web sites is at your own risk.

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:

IBM Corporation
Department T01
Building 062
P.O. Box 12195
RTP, NC 27709-2195
U.S.A.

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 information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement or any equivalent agreement between us.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Trademarks

The following terms are trademarks of the IBM Corporation in the United States, or other countries, or both:

| | |
|--------------------|----------------------|
| CICS | OS/2 |
| eServer | OS/400 |
| i5/OS | Presentation Manager |
| IBM | PS/2 |
| IBM Global Network | System i5 |
| iSeries | VisualAge |
| MVS | zSeries |

Microsoft, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java, JavaBeans, and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Other company, product, and service names may be trademarks or service marks of others.

Index

Numerics

01, Connect Presentation Space 36
02, Disconnect Presentation Space 82
03, Send Key 91, 135, 163, 170
04, Wait 169
05, Copy Presentation Space 57
06, Search Presentation Space 129, 170
07, Query Cursor Location 107
08, Copy Presentation Space to String 64
09, Set Session Parameters 147
10, Query Sessions 112
101, Connect Window Services 37
 Connect Window Services (101) 37
102, Disconnect Window Service 82
103, Query Window Coordinates 115
104, Window Status 170
105, Change Switch List LT Name 33
106, Change PS Window Name 32
11, Reserve 123
110, Start Playing Macro 165
12, Release 123, 124
13, Copy OIA 48
14, Query Field Attribute 107
15, Copy String to Presentation Space 76
16/32 bit considerations 21
18, Pause 100, 168
20, Query System 113
21, Reset System 124, 147
22, Query Session Status 40, 110
23, Start Host Notification 100, 109, 160
24, Query Host Update 101, 109, 168
25, Stop Host Notification 168
30, Search Field 125, 170
31, Find Field Position 42, 86
32-bit presentation space IDs 11
32, Find Field Length 42, 85
3270 terminal emulation 326
33, Copy String to Field 72
34, Copy Field to String 40
40, Set Cursor 146
41, Start Close Intercept 156
42, Query Close Intercept 104
43, Stop Close Intercept 166
45, Query Additional Field Attribute 103
50, Start Keystroke Intercept 163
51, Get Key 89, 102, 163
52, Post Intercept Status 102, 163
53, Stop Keystroke Intercept 168
61, Lock PMSVC API 99
90, Send File 133
91, Receive File 121
99, Convert Position or Convert RowCol 38

A

Allocate Communications Buffer (123) 30
applications
 error codes 349
 using SRPI 328

ASCII Mnemonics
 general 17
 get key (51) function 19
 send key (3) function 19
Asynchronous Functions, WinHLLAPI 179
ATTRB 149
attribute bytes 40, 57, 64, 72, 77, 149
automation 24
AUTORESET 151

B

beep 102
BLANK 152
Blocking Routines 187
buffer size 329

C

C language
 init _ send _ req _ parms 330
 interface 329
 record definition 330
 requesters 329
 send _ request 330
 syntax 329
call (input) parameters
 general 28
call/return 324
calls, prerequisite 28
Cancel File Transfer (92) 31
Change Menu Item 275, 411
Change PS Window Name (106) 32
Change Switch List LT Name (105) 33
character, escape 18, 90, 136, 150
characters, ASCII 18
Code Conversion 218
communication services functions
 Receive File (91) 121
 Send File (90) 133
Communications Manager
 applications 328
compiling and linking 9, 10
Connect for Structured Fields (120) 34
Connect Presentation Space (1)
 functions where not required 36
 general 36
 interaction with disconnect 12
Connect Window Services (101) 37
Convert Position or Convert RowCol (99) 38
Copy Field to String (34) 24, 40
copy functions
 Copy Field to String (34) 40
 Copy OIA (13) 48
 Copy Presentation Space (5) 57
 Copy Presentation Space to String (8) 64
 Copy String to Field (33) 72
 Copy String to Presentation Space (15) 76
Copy OIA (13) 22, 48
Copy Presentation Space (5) 57

- Copy Presentation Space to String (8) 23, 64
- Copy String to Field (33) 24, 72
- Copy String to Presentation Space (15) 76
- CPRB (Connectivity Programming Request Block)
 - storage 329
- create menu item 281, 416
- critical sections 2
- cursor movement 23

D

- data structures 7
- DDE data items
 - LU Topic 216
 - Session Topic 216
 - System Topic 216
- DDE data items in a 16-bit environment 357
- DDE data items, 16-bit
 - LU Topic 358
 - Session Topic 358
 - System Topic 358
- DDE Functions with a DDE Client Application
 - DDE Functions for Session Conversation 298
 - DDE Functions for Session Conversation (Hot Link) 307
 - DDE Functions for System Conversation 295
 - Personal Communications DDE Interface 291, 295
 - Visual Basic Sample Program 311
- DDE functions, 16-bit environment 357
 - Find Field 360
 - function list 358
 - Get Keystrokes 361
 - Get Mouse Input 362
 - Get Number of Close Requests 365
 - Get Operator Information Area 366
 - Get Partial Presentation Space 367
 - Get Presentation Space 369
 - Get Session Status 371
 - Get System Configuration 372
 - naming conventions for parameters 359
 - Set Cursor Position 388
 - Set Mouse Intercept Condition 390
 - Set Presentation Space Service Condition 392
 - Set Session Advise Condition 393
 - Set Structured Field Service Condition 394
 - Start Close Intercept 395
 - Start Keystroke Intercept 397
 - Start Mouse Input Intercept 398
 - Start Read SF 401
 - Start Session Advise 402
 - Stop Close Intercept 403
 - Stop Keystroke Intercept 404
 - Stop Mouse Input Intercept 405
 - Stop Read SF 405
 - Stop Session Advise 406
 - summary of DDE functions in a 16-bit environment 420
 - Terminate Session Conversation 407
 - Terminate Structured Field Conversation 407
 - Terminate System Conversation 407
 - Write SF 408
- DDE functions, 32-bit environment 215
 - Code Conversion 218
 - DDE data items, Windows 32-bit
 - general 215
 - LU Topic 216
 - Session Topic 216
 - System Topic 216
 - Find Field 220

- DDE functions, 32-bit environment (*continued*)
 - function list 216
 - Get Keystrokes 222
 - Get Mouse Input 223
 - Get Number of Close Requests 226
 - Get Operator Information Area 227
 - Get Partial Presentation Space 228
 - Get Presentation Space 230
 - Get Session Status 232
 - Get System Configuration 234
 - Get System Formats 235
 - Get System Status 236
 - Get System SysItems 237
 - Get System Topics 238
 - Get Trim Rectangle 238
 - Initiate Session Conversation 239
 - Initiate Structured Field Conversation 240
 - Initiate System Conversation 240
 - naming conventions for parameters 217
 - Put Data to Presentation Space 241
 - Search for String 242
 - Send Keystrokes 243
 - Session Execute Macro 244
 - Set Cursor Position 251
 - Set Mouse Intercept Condition 253
 - Set Presentation Space Service Condition 255
 - Set Session Advise Condition 257
 - Set Structured Field Service Condition 258
 - Start Close Intercept 259
 - Start Keystroke Intercept 260
 - Start Mouse Input Intercept 261
 - Start Read SF 264
 - Start Session Advise 266
 - Stop Close Intercept 267
 - Stop Keystroke Intercept 268
 - Stop Mouse Input Intercept 268
 - Stop Read SF 269
 - Stop Session Advise 270
 - Terminate Session Conversation 270
 - Terminate Structured Field Conversation 271
 - Terminate System Conversation 271
 - Write SF 272
- DDE menu functions, 16-bit environment 409
 - Change Menu Item 411
 - create menu item 416
 - Initiate Menu Conversation 417
 - list 411
 - Start Menu Advise 418
 - Stop Menu Advise 419
 - Terminate Menu Conversation 420
- DDE menu functions, 32-bit environment 273
 - Change Menu Item 275
 - create menu item 281
 - Initiate Menu Conversation 282
 - list 274
 - Start Menu Advise 282
 - Stop Menu Advise 284
 - Terminate Menu Conversation 284
- debugging 19
- default, values 328
- device services functions
 - Get Key (51) 89
 - Post Intercept Status (52) 102
 - Release (12) 123
 - Reserve (11) 123
 - Start Keystroke Intercept (50) 163
 - Stop Keystroke Intercept (53) 168

- directory, default
 - Receive File 123
- Disconnect from Structured Fields (121) 80
- Disconnect Presentation Space (2)
 - general 82
 - interaction with connect 12
- Disconnect Window Service (102) 82
- dynamic link method 10

E

- EAB 152
- Edit keys
 - intercepting 83
- EditKey Intercept 83
- EHLLAPI
 - functions 27
 - summary 28
- EHLLAPI call format 6
- EHLLAPI Overviews
 - IBM Enhanced EHLLAPI vs. IBM Standard EHLLAPI 6
 - IBM Standard EHLLAPI 5
 - WinHLLAPI 5
 - WinHLLAPI vs. IBM Standard EHLLAPI 5
- EHLLAPI programming overview 5
- EHLLAPI return codes 8
- EOT 149
- error handling 349
- ESC 150
- escape character 18, 90, 136, 150
- exception code values 354
- exception object values 354

F

- field-formatted PS 42, 125
- field-related functions
 - Copy Field to String (34) 40
 - Copy String to Field (33) 72
 - Find Field Length (32) 85
 - Find Field Position (31) 86
 - Query Additional Field Attribute (45) 103
 - Query Field Attribute (14) 107
 - Search Field (30) 125
- fields, host
 - input protected 135
 - numeric only 135
- file transfer 24
- file transfer functions
 - Receive File (91) 121
 - Send File (90) 133
- Find Field 220, 360
- Find Field Length (32) 42, 85
- Find Field Position (31) 42, 86
- flow, requester and server 326
- FPAUSE 150
- Free Communications Buffer (124) 88
- function calls
 - call (input) parameters 28
 - notes on using the function 28
 - page layout conventions 27
 - prerequisite calls 28
 - return (output) parameters 28
 - use of 27

G

- Get Key (51) 17, 89, 102, 163
- Get Keystrokes 222, 361
- Get Mouse Input 223, 362
- Get Number of Close Requests 226, 365
- Get Operator Information Area 227, 366
- Get Partial Presentation Space 228, 367
- Get Presentation Space 230, 369
- Get Request Completion (125) 94
- Get Session Status 232, 371
- Get System Configuration 234, 372
- Get System Formats 235, 373
- Get System Status 236, 374
- Get System SysItems 237, 375
- Get System Topics 238, 376
- Get Trim Rectangle 238, 377

H

- Hindi, code page 1137
 - Convert Position of Convert RowCol (99) 40
 - Copy Field to String (34) 47
 - Copy Presentation Space (5) 63
 - Copy Presentation Space to String (8) 71
 - Copy String to Field (33) 75
 - Copy String to Presentation Space (15) 79
 - Get Key (51) 93
 - Search Field (30) 128
 - Search Presentation Space (6) 132
 - Send Key (3) 145
 - Set Cursor (40) 147
 - Set Session Parameters (9) 155
- host
 - computer router 325
 - computer server 325
- host automation scenarios 22
- host fields
 - input protected 135
 - numeric only 135
- host-connected presentation space 12

I

- IBM Support Center 114
- init _ send _ req _ parms
 - C language 330
- Initialization/Termination Functions 186
- Initiate Menu Conversation 282, 417
- Initiate Session Conversation 239, 377
- Initiate Structured Field Conversation 240, 378
- Initiate System Conversation 240, 379
- input protected fields 135
- 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 329
- IPAUSE 150

J

- Japanese, code page 1390/1399
 - Copy Field to String (34) 46

Japanese, code page 1390/1399 (*continued*)
Copy Presentation Space (5) 62
Copy Presentation Space to String (8) 70
Copy String to Field (33) 74
Copy String to Presentation Space (15) 78
Get Key (51) 91
Search Field (30) 127
Search Presentation Space (6) 131
Send Key (3) 145
Set Session Parameters (9) 155

K

keyboard enhancement 25
keyboard mnemonics
 general 17
 tables 137
keyboard, session 17
keystroke filtering 25
keystroke interception, Get Key (51) 89

L

language interface
 C language 329
languages 6
Linking
 description 9
 Dynamic Link Method 10
 Static Link Method 10
Lock Presentation Space API (60) 97
Lock Window Services API (61) 99
locking presentation space 16
LWAIT 151, 170

M

memory allocation 8
mnemonics
 ASCII 18
 for Send Key 17
 keyboard, tables 137
 shift key 17
Multithreading 11

N

NOATTRB 149
NOBLANK 152
NOEAB 152
NOQUIET 150
NORESET 151
NOXLATE 152
NULLATTRB 149
numeric only fields 135
NWAIT 151, 170

O

OIA 48, 170
Operator Information Area
 See "OIA." 48
operator services functions
 Pause (18) 100
 Query Host Update (24) 109

operator services functions (*continued*)
 Query Session Status (22) 110
 Query Sessions (10) 112
 Query System (20) 113
 Reset System (21) 124
 Send Key (3) 135
 Set Session Parameters (9) 147
 Start Host Notification (23) 160
 Stop Host Notification (25) 168
 Wait (4) 169
options 155

P

parameters
 call 28
 returned 328
 SEND _ REQUEST 326
 supplied 326
path, default
 Receive File 123
 Send File 135
Pause (18) 23, 100, 168
PCSAPI
 general 191
 how to use 191
 pcsConnectSession 191
 pcsDisconnectSession 192
 pcsGetPageSettings 199
 pcsGetPrinterSettings 205
 pcsQueryConnectionInfo 193
 pcsQueryEmulatorStatus 194
 pcsQuerySessionList 194
 pcsQueryWorkstationProfile 196
 pcsRestorePageDefaults 201
 pcsSetLinkTimeout 197
 pcsSetPageSettings 202
 pcsSetPrinterSettings 210
 pcsStartSession 197
 pcsStopSession 198
pcsDisconnectSession 192
pcsQueryConnectionInfo 193
pcsQueryEmulatorStatus 194
pcsQuerySessionList 194
pcsQueryWorkstationProfile 196
pcsStartSession 197
pcsStopSession 198
performance considerations 329
Post Intercept Status (52) 26, 102, 163
prerequisite calls, general 28
presentation services functions
 Connect Presentation Space (1) 36
 Copy Field to String (34) 40
 Copy OIA (13) 48
 Copy Presentation Space (5) 57
 Copy Presentation Space to String (8) 64
 Copy String to Field (33) 72
 Copy String to Presentation Space (15) 76
 Disconnect Presentation Space (2) 82
 Find Field Length (32) 85
 Find Field Position (31) 86
 Get Request Completion (125) 94
 Lock Presentation space API (60) 97
 Query Additional Field Attribute (45) 103
 Query Cursor Location (7) 107
 Query Field Attribute (14) 107
 Search Field (30) 125

- presentation services functions (*continued*)
 - Search Presentation Space (6) 129
 - Set Cursor (40) 146
- presentation space
 - character table 49
 - cursor movement 23
 - Enhanced 32-bit interface 11
 - field-formatted 40, 42, 72, 85, 86, 125
 - host-connected 12
 - how specified 12
 - identifier
 - blank specifier 13
 - function 12
 - how processed 12
 - letter specifier 13
 - null specifier 13
 - processing for functions not requiring connect 13
 - processing for functions requiring connect 13
 - OIA 48
 - types 11
- presentation space names
 - declaring 12
 - maximum number of 12
 - valid names 12
- presentation spaces 11
- programming interface, server-requester 324
- PSID handling
 - functions not requiring connect 13
 - functions requiring connect 13
- Put Data to Presentation Space 241, 379

Q

- Query Additional Field Attribute (45) 103
- Query Close Intercept (42) 104
- Query Communication Event (81) 106
- Query Communications Buffer Size (122) 104
- Query Cursor Location (7) 107
- Query Field Attribute (14) 107
- Query Host Update (24) 101, 109, 160, 168
- Query Reply Data Structures Supported by EHLLAPI
 - Architecture Query Reply 340
 - Cooperative Processing Requester Query Reply 338
 - general 331
 - IBM Auxiliary Device Query Reply
 - Direct Access Self-Defining Parameter 335
 - general 334
 - PCLK Protocol Controls Self-Defining Parameter 336
 - OEM Auxiliary Device Query Reply
 - Direct Access Self-Defining Parameter 337
 - general 336
 - PCLK Protocol Controls Self-Defining Parameter 337
 - Product-Defined Query Reply
 - Direct Access Self-Defining Parameter 339, 341
 - general 338
 - Optional Parameters 338
 - The DDM Query Reply
 - Base DDM Query Reply Formats 332
 - DDM Application Name Self-Defining Parameter 332
 - general 331
 - PCLK protocol controls Self-Defining Parameter 332
- Query Session Status (22) 40, 110
- Query Sessions (10) 112
- Query System (20) 113
- Query Window Coordinates (103) 115
- Query_Emulator_Status 453
- Query_Session_List 456

- Query_Workstation_Profile 461
- QUIET 150

R

- Read Structured Fields (126) 116
- Receive File (91)
 - default path for target file 123
 - general 24, 121, 150
- RECEIVE.EXE location 121
- relationship requester server 325
- Release (12) 24, 123, 124
- requester
 - C language 329
 - server flow, and 326
 - server relationship 325
- Reserve (11) 24, 123
- Reset System (21) 123, 124, 147
- return (output) parameters, general 28
- return codes 349
- REXX EHLLAPI functions
 - Query_Emulator_Status 453
 - Query_Session_List 456
 - Query_Workstation_Profile 461
 - Send_File 468
 - Sendkey 469
 - Set_Cursor_Pos 470
 - Set_Session_Parms 471
 - Set_Window_Status 472
 - Start_Close_Intercept 473
 - Start_Communication 474
 - Start_Host_Notify 475
 - Start_Keystroke_Intercept 476
 - Start_Session 477
 - Stop_Close_Intercept 478
 - Stop_Communication 479
 - Stop_Host_Notify 480
 - Stop_Keystroke_Intercept 481
 - Stop_Session 482
 - Wait 483
- REXX EHLLAPI programming, notes 484
- router, SRPI 325, 326

S

- sample program, a simple EHLLAPI 19
- sample programs 3
- Search Field (30) 125, 170
- Search for String 242, 380
- search functions 22
 - Search Field (30) 125
 - Search Presentation Space (6) 129
- Search Presentation Space (6) 23, 129, 170
- SEND _ REQUEST
 - invoking 329
 - parameters
 - returned 328
 - supplied 326
 - processing errors 349
 - routing 325
- send _ request function
 - C language 330
- Send File (90)
 - default path for target file 135
 - general 24, 133, 150
 - SEND.EXE location 133

- Send Key (3) 17, 91, 135, 163, 170
- Send Keystrokes 243, 381
- Send_File 468
- sending keystrokes 23
 - mnemonics 17
 - Send Key (3) 135
- Sendkey 469
- server
 - name 328
 - return codes 355
- server-requester programming interface 324
- service 114
- Session Execute Macro 244, 382
- session keyboard 17
- Set Cursor (40) 146
- Set Cursor Position 251, 388
- Set Mouse Intercept Condition 253, 390
- Set Presentation Space Service Condition 255, 392
- Set Session Advise Condition 257, 393
- Set Session Parameters (9)
 - general 136, 147
 - List of affected functions 147
 - string specification 148
 - Valid Input 148, 155
- Set Structured Field Service Condition 258, 394
- Set_Cursor_Pos 470
- Set_Session_Parms 471
- Set_Window_Status 472
- shift key mnemonics 17
- size of presentation spaces 11
- source code syntax 21
- specifying strings 77
- SRCHALL 149
- SRCHBKWD 149
- SRCHFROM 149
- SRCHFRWD 149
- stack size 2
- Start Close Intercept 259, 395
- Start Close Intercept (41) 156
- Start Communication Notification (80) 158
- Start Host Notification (23) 100, 109, 150, 160, 168
- Start Keystroke Intercept 260, 397
- Start Keystroke Intercept (50) 163
- Start Menu Advise 282, 418
- Start Mouse Input Intercept 261, 398
- Start Playing Macro (110) 165
- Start Read SF 264, 401
- Start Session Advise 266, 402
- Start_Close_Intercept 473
- Start_Communication 474
- Start_Host_Notify 475
- Start_Keystroke_Intercept 476
- Start_Session 477
- static link method 10
- Stop Close Intercept 267, 403
- Stop Close Intercept (43) 166
- Stop Communication Notification (82) 167
- Stop Host Notification (25) 168
- Stop Keystroke Intercept 268, 404
- Stop Keystroke Intercept (53) 168
- Stop Keystroke Intercept (53), you can call the 26
- Stop Menu Advise 284, 419
- Stop Mouse Input Intercept 268, 405
- Stop Read SF 269, 405
- Stop Session Advise 270, 406
- Stop_Close_Intercept 478
- Stop_Communication 479

- Stop_Host_Notify 480
- Stop_Keystroke_Intercept 481
- Stop_Session 482
- STREOT 149
- string interception, Get Key (51) 89
- string specification
 - session options 148
- STRLEN 149
- syntax, C language 329

T

- Terminate Menu Conversation 284, 420
- Terminate Session Conversation 270, 407
- Terminate Structured Field Conversation 271, 407
- Terminate System Conversation 271, 407
- TIMEOUT 150
- trademarks 486
- transport layer errors 349
- TWAIT 151, 170
- types of presentation spaces 11

U

- UERCPRB, C language 330
- Unicode
 - Hindi, code page 1137
 - Convert Position of Convert RowCol (99) 40
 - Copy Field to String (34) 47
 - Copy Presentation Space (5) 63
 - Copy Presentation Space to String (8) 71
 - Copy String to Field (33) 75
 - Copy String to Presentation Space (15) 79
 - Get Key (51) 93
 - Search Field (30) 128
 - Search Presentation Space (6) 132
 - Send Key (3) 145
 - Set Cursor (40) 147
 - Set Session Parameters (9) 155
 - Japanese, code page 1390/1399
 - Copy Field to String (34) 46
 - Copy Presentation Space (5) 62
 - Copy Presentation Space to String (8) 70
 - Copy String to Field (33) 74
 - Copy String to Presentation Space (15) 78
 - Get Key (51) 91
 - Search Field (30) 127
 - Search Presentation Space (6) 131
 - Send Key (3) 145
 - Set Session Parameters (9) 155
- using API header files 2

W

- Wait 483
- Wait (4) 22, 136, 169
- window services functions
 - Change PS Window Name (106) 32
 - Change Switch List LT Name (105) 33
 - Lock Window Services API (61) 99
- Window Status (104) 170
- WinHLLAPI Extension Functions
 - Asynchronous Functions
 - general 179
 - WinHLLAPIAsync 179
 - WinHLLAPICancelAsyncRequest 185

WinHLLAPI Extension Functions (*continued*)

Blocking Routines

general 187

WinHLLAPICancelBlockingCall 188

WinHLLAPIIsBlocking 187

WinHLLAPISetBlockingHook 187

WinHLLAPIUnhookBlockingHook 188

general 186

Initialization/Termination Functions

general 186

WinHLLAPI Cleanup 187

WinHLLAPI Startup 186

Summary 179

Write SF 272, 408

Write Structured Fields (127) 173

X

XLATE 152



Program Number: 5639-I70

Printed in USA

SC31-8478-10

